Actividades

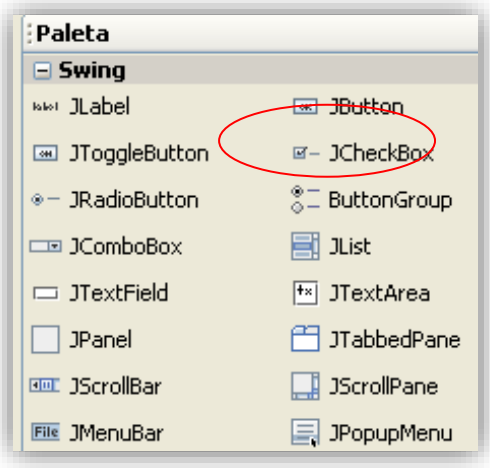
Estos ejercicios son para la práctica de estudio.

Son Ejercicios guiados paso a paso. Con enlaces a videos en YouTube.-

Ejercicios

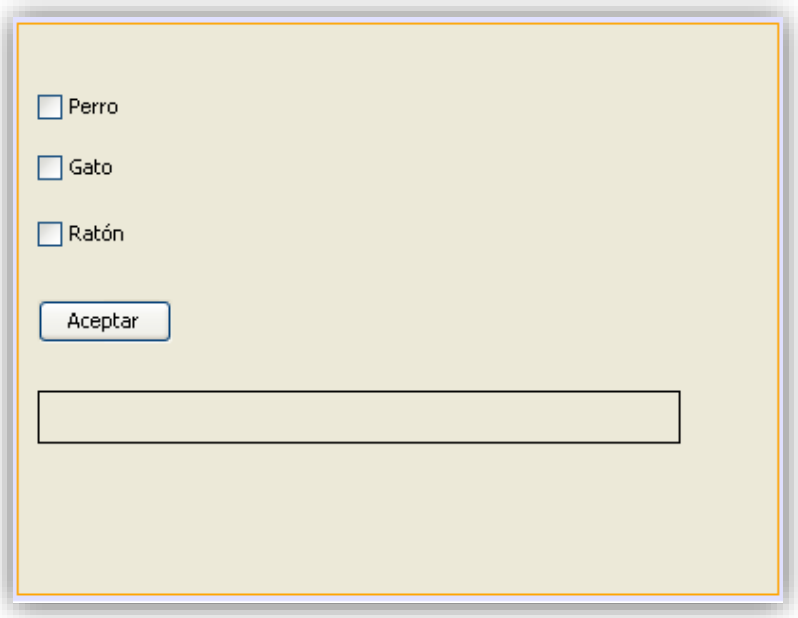
Ejercicio.S2.3.1 Cuadros de verificación (JCheckBox))
Video en You Tube (U3S2E1) : https://www.youtube.com/watch?v=TKF7pz3NZUk

1. Realiza un nuevo proyecto.
2. En la ventana principal debes añadir lo siguiente:
   a. Un botón “Aceptar” llamado btnAceptar.
   b. Una etiqueta con borde llamada etiResultado.
3. Añade también tres cuadros de verificación. Estos cuadros son objetos del tipo JCheckBox.



4. Añade tres JCheckBox y cambia el texto de ellos, de forma que aparezca “Perro”, “Gato” y “Ratón”.
5. Debe cambiar el nombre de cada uno de ellos. Se llamarán: chkPerro, chkGato, chkRaton.

6. La ventana tendrá el siguiente aspecto cuando termine:



7. El programa debe funcionar de la siguiente forma:
Cuando el usuario pulse aceptar, en la etiqueta aparecerá un mensaje indicando qué animales han sido "seleccionados". Para ello hay que programar el evento ***actionPerformed*** del botón Aceptar.
Es necesario adelantarnos explicando la estructura de control condicional **if**:
La estructura condicionar **"if"** permite tomar decisiones en un algoritmo o en un programa. La sintaxis que deberemos emplear en Java es la siguiente:

```
if <expresión booleana>
{ sentencias 1, por verdadero }
else {sentencias 2, por falso };
```

Esta sentencia indica al compilador que la expresión boolean que colocamos después del **If** arroja un valor verdadero (True), se debe ejecutar el bloque de sentencias 1; en caso contrario ejecutará el bloque de sentencias 2 .

En el evento ***actionPerformed*** del botón Aceptar añada el siguiente código:

```
String mensaje="Animales elegidos: ";
if (chkPerro.isSelected()) {
    mensaje=mensaje+"Perro ";}
if (chkGato.isSelected()) {
    mensaje=mensaje+"Gato ";}
if (chkRaton.isSelected()) {
    mensaje=mensaje+"Raton ";}
etiResultado.setText(mensaje);
```

8. Observa el código. En él se hace lo siguiente:

    a. Se crea una variable de cadena llamada *mensaje*.
    b. En esa variable se introduce el texto “Animales elegidos: “
    c. Luego, compruebo si está seleccionada la casilla de verificación chkPerro. Si es así concateno a la cadena mensaje la palabra “Perro”.
    d. Luego compruebo si está seleccionada la casilla de verificación chkGato y hago lo mismo.
    e. Lo mismo con la casilla chkRaton.
    f. Finalmente presento la cadena mensaje en la etiqueta etiResultado.

9. Observa el método **isSelected()** propio de las casillas de verificación, permiten saber si una casilla está activada o no.

10. Ejecute el programa. Seleccione por ejemplo las casillas Gato y Ratón. Al pulsar Aceptar el resultado debe ser el siguiente:

CONCLUSIÓN

Los cuadros de verificación (JCheckBox) se usan cuando quieres seleccionar varias opciones.

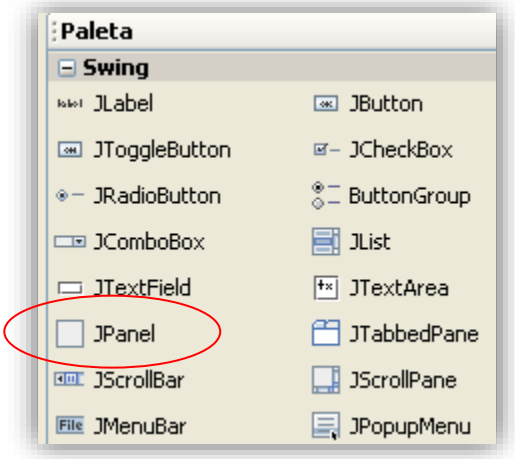
Ejercicio.3.S2.2.
Video en You Tube (U3S2-E02) : https://www.youtube.com/watch?v=mExLEOPHGyE

1. Realiza un nuevo proyecto.

2. En la ventana principal debes añadir lo siguiente:

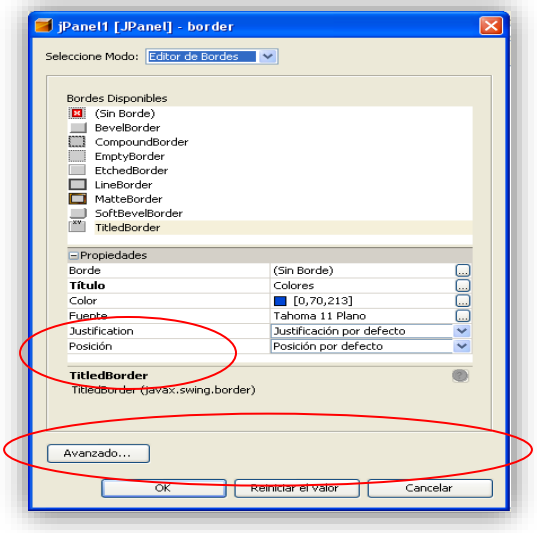
   a. Un botón “Aceptar” llamado btnAceptar.
   b. Una etiqueta con borde llamada etiResultado.

3. Añade un panel. Un panel es una zona rectangular que puede contener elementos (botones, etiquetas, etc) La forma de poner un panel es a través del objeto JPanel.

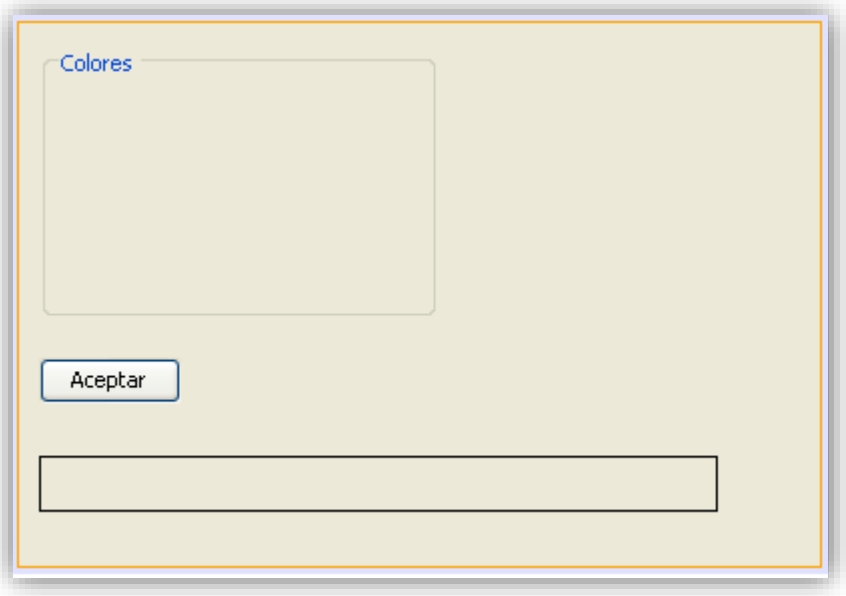


4. Una vez añadido el panel en el JFrame, le pondremos un borde para poder localizarlo fácilmente. Debes hacer lo siguiente:

   a. Selecciona el panel que has añadido.
   b. Activa la propiedad Border (botón con tres puntos)
   c. Busca el tipo de borde llamado TitledBorder (borde con título) y pon el título colores.

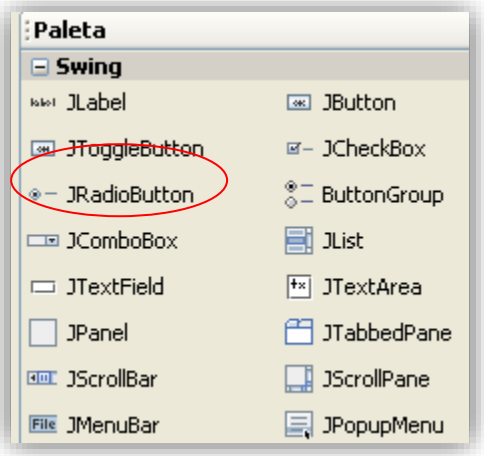


5. Tu ventana debe quedar más o menos así:



6. Ahora debes añadir tres botones de opción (botones de radio) dentro del panel. Estos botones son objetos del tipo **JRadioButton**.

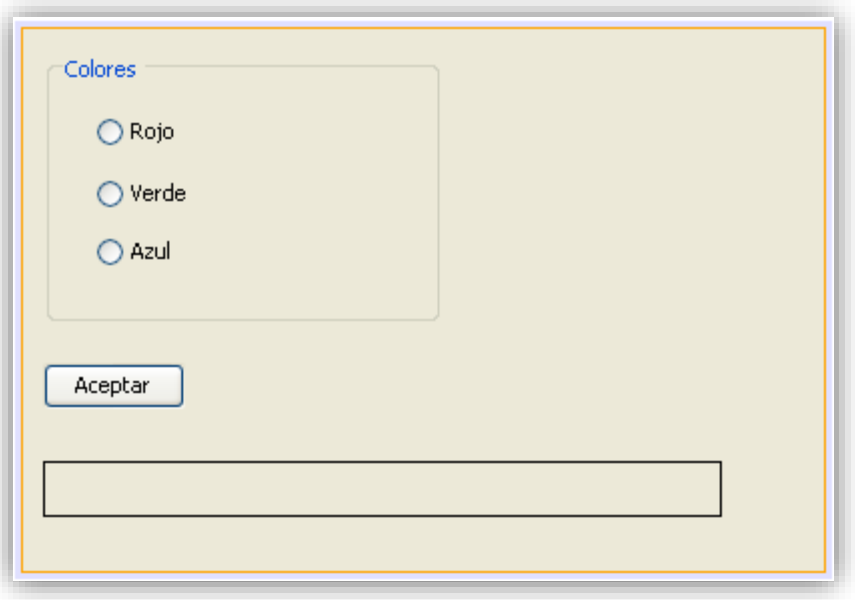
7.



8. Añade tres **JRadioButton** y cambia el texto de ellos, de forma que aparezca “Rojo”, “Verde” y “Azul”.

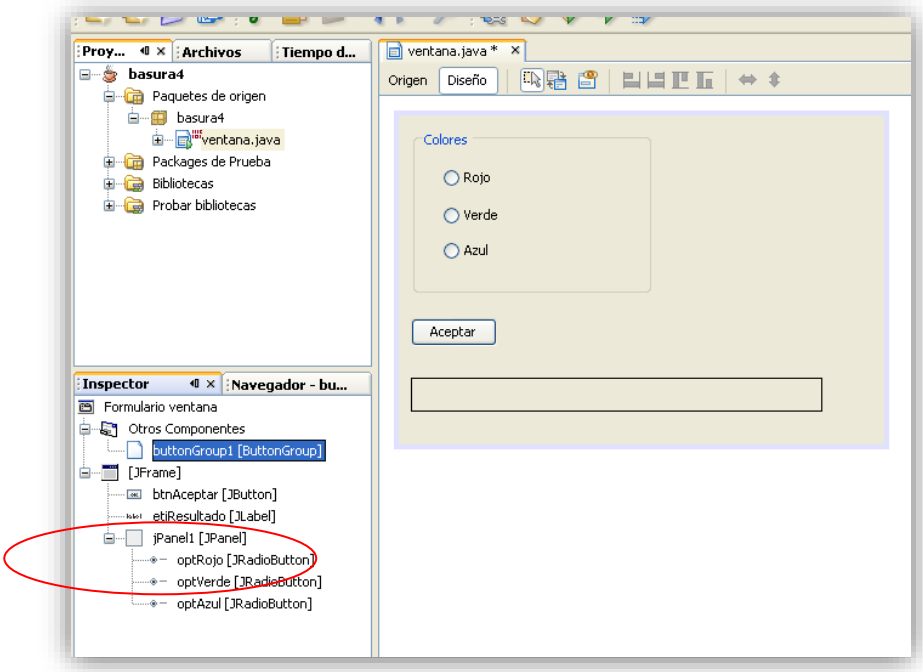
9. Debe cambiar el nombre de cada uno de ellos. Se llamarán: optRojo, optVerde, optAzul.

10. La ventana tendrá el siguiente aspecto cuando termine:

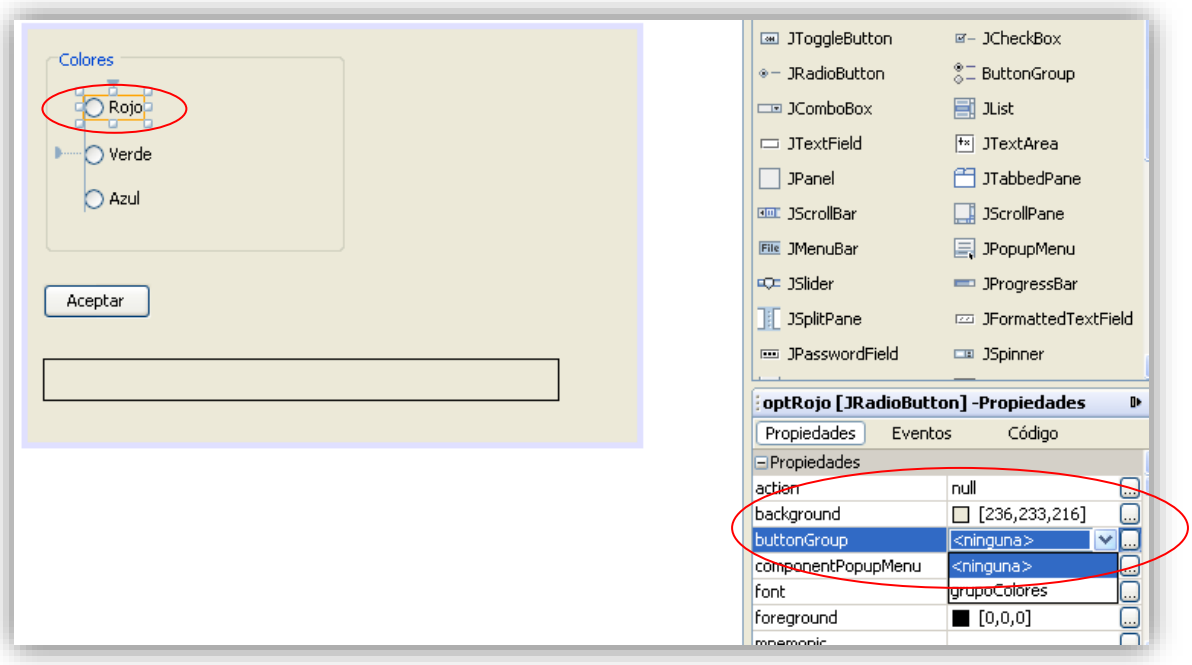


11. Si ejecuta el programa, observará que pueden seleccionarse varios colores a la vez. Esto no es interesante, ya que los botones de opción se usan para activar solo una opción entre varias.
12. Hay que hacer que solo un botón de opción pueda estar seleccionado a la vez. Para ello, debe añadir un nuevo objeto. Realice los siguientes pasos:

    a. Añada un objeto del tipo **ButtonGroup** al formulario. ¡Atención! Este objeto es invisible, y no se verá en el formulario, sin embargo, lo podréis ver en el Inspector, en la parte de “Otros Componentes”:



b. Tienes que darle un nombre al **ButtonGroup**:
   El nombre será **"grupoColores"**.
c. Ahora, hay que conseguir que los tres botones pertenezcan al mismo grupo. Es decir, que pertenezcan al grupo **grupoColores**.
d. Selecciona el botón de opción optRojo y cambia su propiedad buttonGroup, indicando que pertenece al grupo colores (observa la imagen):



e. Haz lo mismo con los botones optVerde y optAzul.

13. Acabas de asociar los tres botones de opción a un mismo grupo. Esto produce que solo una de las tres opciones pueda estar activada. Pruébelo ejecutando el programa.

14. Ahora interesa que la opción "Rojo" salga activada desde el principio. Una forma de hacer esto es programando en el "Constructor" lo siguiente:

```
optRojo.setSelected(true);
```

El método **setSelected** hace que se pueda activar o desactivar un botón de opción.

Prueba el programa. Observa como la opción Rojo está activada inicialmente.

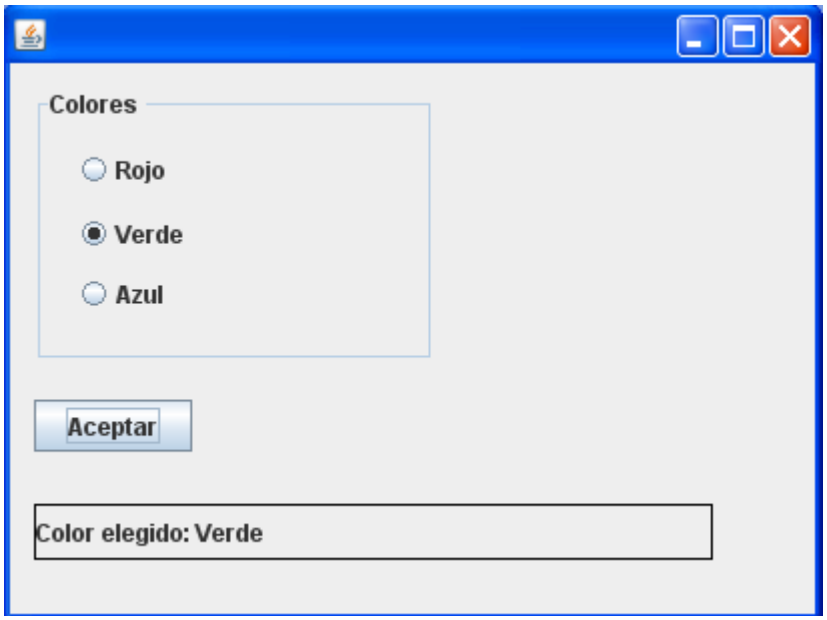
15. El programa no está terminado aún. Interesa que cuando el usuario pulse el botón Aceptar, en la etiqueta aparezca el color elegido. Para ello, en el ***actionPerformed*** del botón Aceptar programe lo siguiente:

```
String mensaje="Color elegido: ";

if (optRojo.isSelected()) {
    mensaje=mensaje+"Rojo";
} else if (optVerde.isSelected()) {
    mensaje=mensaje+"Verde";
} else if (optAzul.isSelected()) {
    mensaje=mensaje+"Azul";
}

etiResultado.setText(mensaje);
```

16. Observa el código. En él se hace lo siguiente:

    a. Se crea una variable de cadena llamada *mensaje*.
    b. En esa variable se introduce el texto "Color elegido: "
    c. Luego se comprueba que opción está seleccionada, usando el método isSelected de los botones de opción. Este método te dice si un botón está seleccionado o no.
    d. Según la opción que esté seleccionada, se añade un texto u otro a la cadena *mensaje*.
    e. Finalmente se muestra la cadena *mensaje* en la etiqueta etiResultado.

17. Ejecute el programa. Seleccione por ejemplo la Verde. Al pulsar Aceptar el resultado debe ser el siguiente:



CONCLUSIÓN

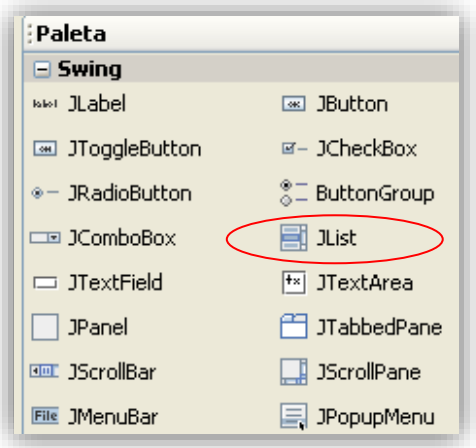
Los botones de opción, también llamados botones de radio (JRadioButton) se usan cuando quieres que el usuario pueda elegir una opción de entre varias.

Es interesante que los botones de radio aparezcan dentro de un panel JPanel. Se recomienda colocar un borde al panel.

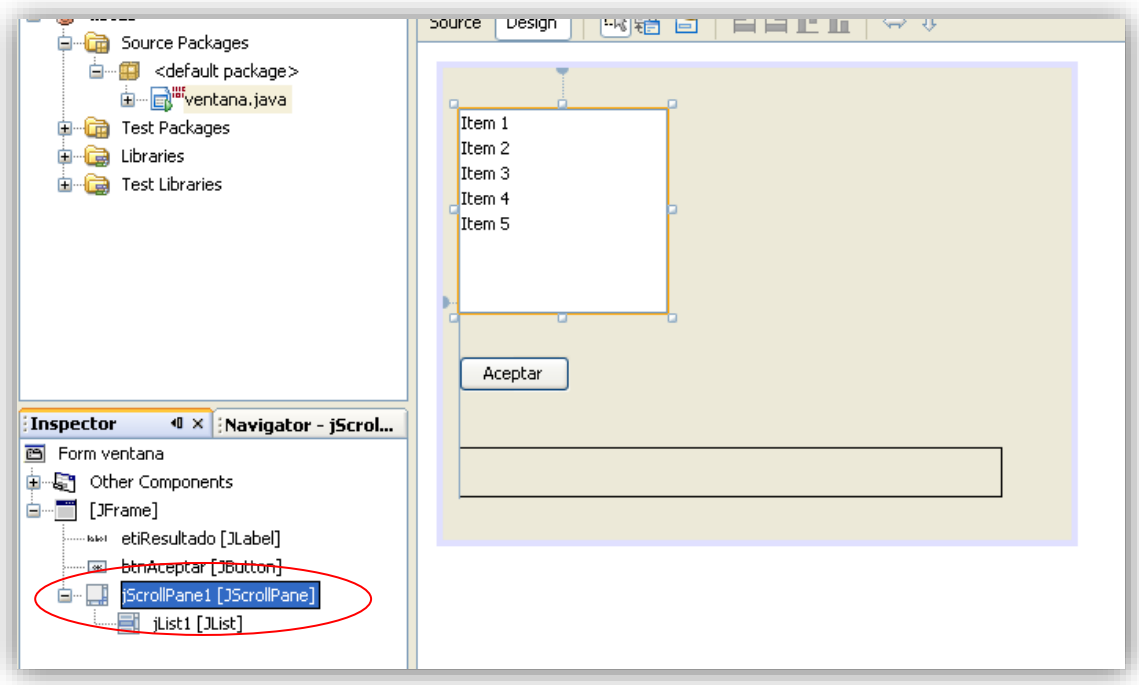
Es totalmente necesario añadir un objeto del tipo ButtonGroup, y hacer que los botones de radio pertenezcan a dicho grupo. En caso contrario, será posible activar varios botones de opción a la vez.

Ejercicio.3.S2.3. Listas (JList)
Video en You Tube (U3S2 E03): https://www.youtube.com/watch?v=GpfcrlSbVko

1. Realiza un nuevo proyecto.
2. En la ventana principal debes añadir lo siguiente:
   a. Un botón “Aceptar” llamado btnAceptar.
   b. Una etiqueta con borde llamada etiResultado.
3. Añade un cuadro de lista. Los cuadros de listas son objetos JList.



4. Cámbiale el nombre al **JList**. Ten cuidado, ya que en los JList aparecen siempre dentro de otro objeto llamado **jScrollPane**. Si miras en el Inspector, verás que al pulsar en el botón + del jScrollPane aparecerá tu JList:

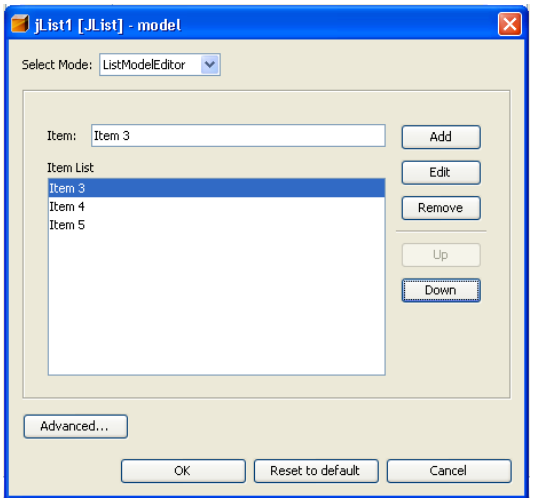


5. Aprovecha para cambiarle el nombre al JList. El nuevo nombre será lstColores.

6. Si te fijas en el JList, consiste en un cuadro que contiene una serie de Items. Estos elementos pueden ser cambiados a través de la propiedad Model del JList.

7. Busca la propiedad Model y haz clic en el botón de los tres puntos. Aparecerá un cuadro de diálogo parecido al siguiente. Solo tienes que seleccionar los elementos que quieras y pulsar el botón “Borrar” (Remove) para eliminarlos de la lista.

8. Puedes añadir elementos escribiéndolos en el cuadro Artículo y luego pulsando el botón “Añadir” (Add).



9. Debes hacer que la lista sea la siguiente:

Rojo
Verde
Azul

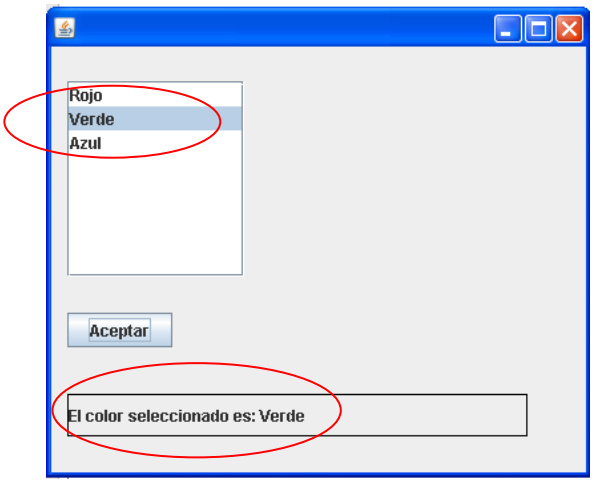
10. Ahora programaremos el *actionPerformed* del botón Aceptar. Debes introducir el siguiente código:

```
String mensaje;

mensaje="El color seleccionado es:" +
lstColores.getSelectedValue().toString();
etiResultado.setText(mensaje);
```

11. Observa el código:

    a. Se crea una variable de cadena llamada *mensaje*.
    b. Y dentro de esta variable se introduce una concatenación de cadenas.
    c. Observa la parte: `lstColores.getSelectedValue()`, esta parte devuelve el valor seleccionado de la lista.
    d. Hay que tener en cuenta que este valor no es una cadena, por eso hay que convertirla a cadena añadiendo `.toString()`.
    e. De esta manera puedes extraer el elemento seleccionado de un cuadro de lista.
    f. Luego simplemente ponemos la cadena mensaje dentro de la etiqueta.

12. Ejecuta el programa y observa su funcionamiento. Por ejemplo, si seleccionas el color verde y pulsas aceptar el resultado será el siguiente:



13. Vamos a mejorar el programa. Puede suceder que el usuario no seleccione ningún valor del cuadro de lista, y sería interesante en este caso que el pro-

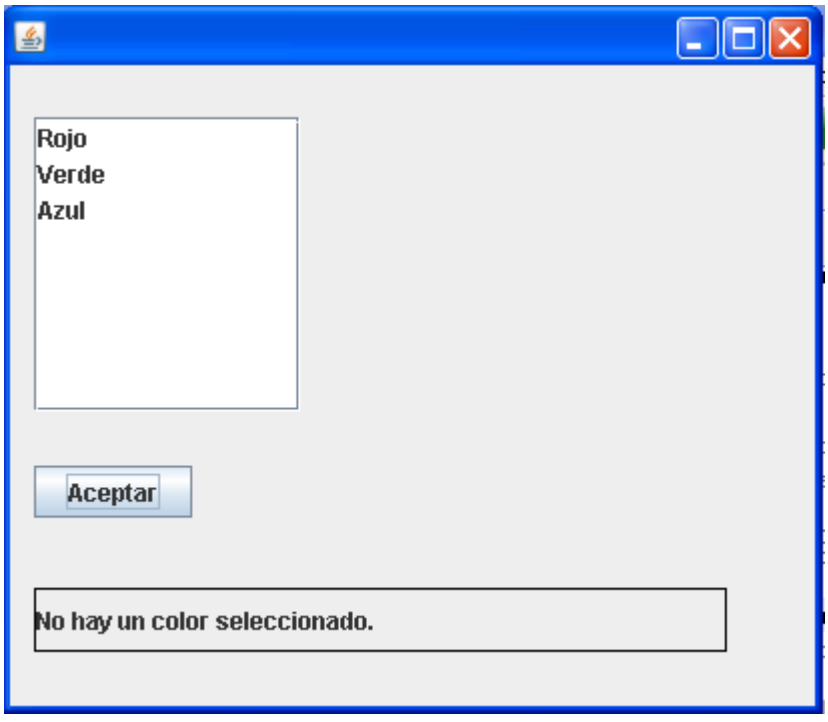
grama avisara de ello. Cambie el código del botón Aceptar por este otro código:

```
String mensaje;

if (lstColores.getSelectedIndex()==-1) {
   mensaje="No hay un color seleccionado.";
} else {
   mensaje="El color seleccionado es:" +lstColores.getSelectedValue().toString();
}
etiResultado.setText(mensaje);
```

14. Observa el código:

    a. El método **getSelectedIndex** me dice el índice del elemento que está seleccionado.
    b. Por ejemplo, si está seleccionado el primero el índice es **0**, si está seleccionado el segundo el índice es 1, etc.
    c. Si este método devuelve **-1**, entonces es señal de que no hay ningún elemento seleccionado.
    d. Aprovecho esto para mostrar un mensaje indicando lo sucedido.

15. Si ejecuta el programa y pulsa el botón Aceptar sin seleccionar nada el resultado debería ser el siguiente:



16. Se podría haber prescindido del botón aceptar si el código anterior se hubiera puesto en el evento **mouseClicked** del cuadro de lista en vez de en el ***actionPerformed*** del botón Aceptar. En este caso, cada vez que se seleccionara un elemento de la lista, automáticamente aparecería el mensaje en la etiqueta.

Se anima a que realice esta modificación.

CONCLUSIÓN

El objeto JList permite crear cuadros de lista. Estos objetos contienen una serie de elementos que pueden ser seleccionados.

A través del método getSelectedValue se puede obtener el elemento que está seleccionado. (Recuerda convertirlo a cadena con toString)

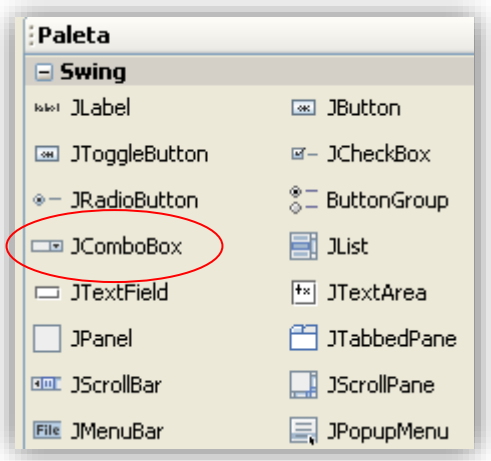
A través del método getSelectedIndex se puede saber la posición del elemento seleccionado. Si este índice es -1, entonces sabremos que no hay ningún elemento seleccionado.

Ejercicio.3.S2.4. CUADROS COMBINADOS (JComboBox)

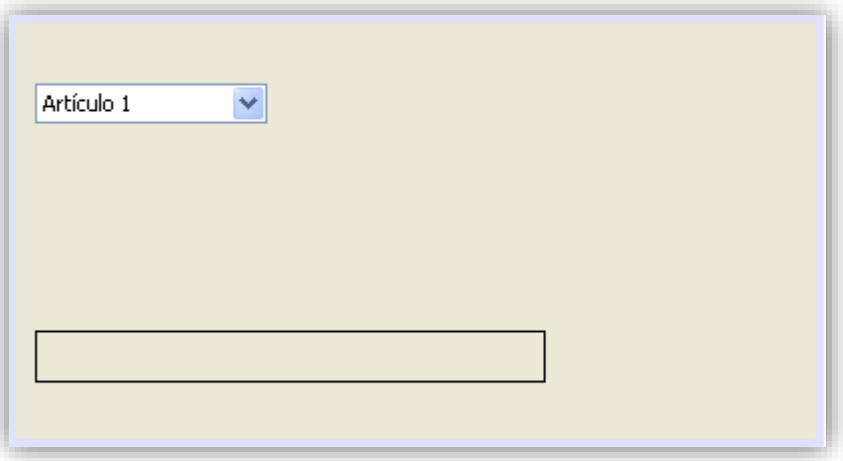
Video en You Tube (U3S2 E04): https://www.youtube.com/watch?v=sPy5Zcwo820

Haz click en el link para verlo

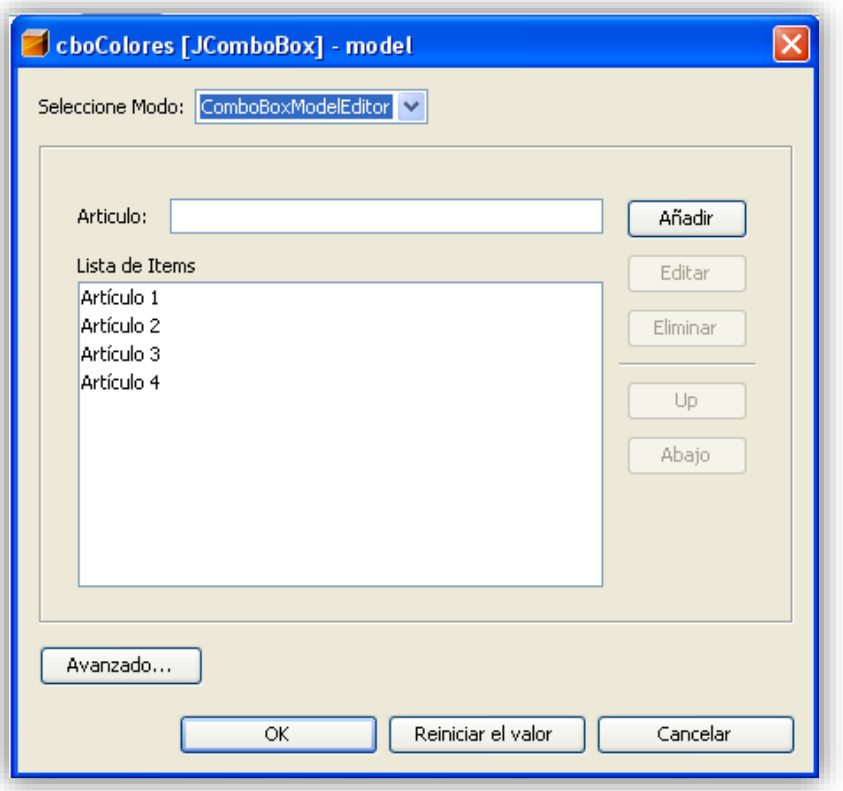
1. Realiza un nuevo proyecto.
2. En la ventana principal debes añadir lo siguiente:
    a. Una etiqueta con borde llamada etiResultado.
3. Añade un cuadro combinado (combo). Los cuadros combinados son objetos del tipo JComboBox. Básicamente, un combo es una lista desplegable.



4. Cámbiale el nombre al **JComboBox**. El nombre será cboColores. Tu programa debe tener más o menos este aspecto.



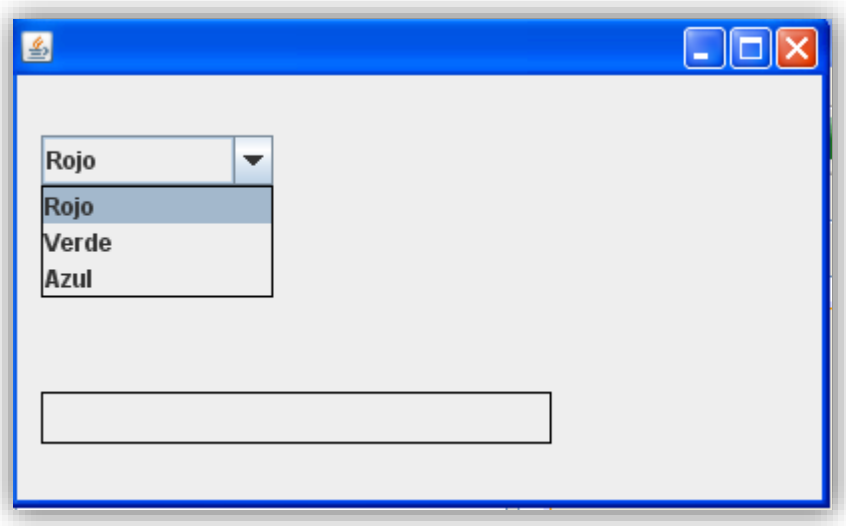
5. Los elementos del cboColores pueden ser cambiados a través de la propiedad **Model**. Selecciona el combo y activa la propiedad **Model** (el botoncito con los tres puntos) Aparecerá lo siguiente:



6. Al igual que pasaba con los cuadros de lista, se pueden eliminar los elementos que contiene el combo y añadir elementos propios. Use los botones Añadir y Eliminar para añadir la siguiente lista de elementos:

Rojo Verde Azul

7. Ejecuta el programa y observa el funcionamiento del desplegable...



8. Vamos a hacer que cuando se elija un elemento del desplegable, en la etiqueta aparezca un mensaje indicando el color elegido.

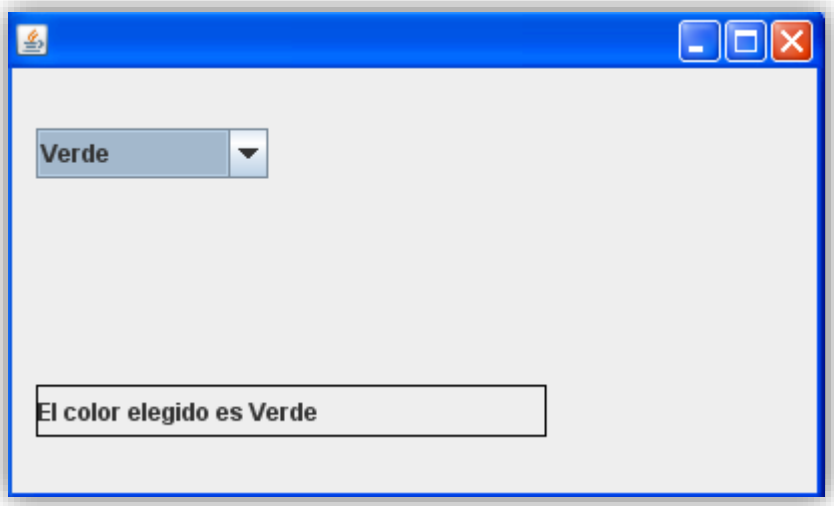
Para ello, debes programar el evento ***actionPerformed*** del combo y añadir el siguiente código:

```
String mensaje="El color elegido es ";

mensaje= mensaje+cboColores.getSelectedItem().toString();
etiResultado.setText(mensaje);
```

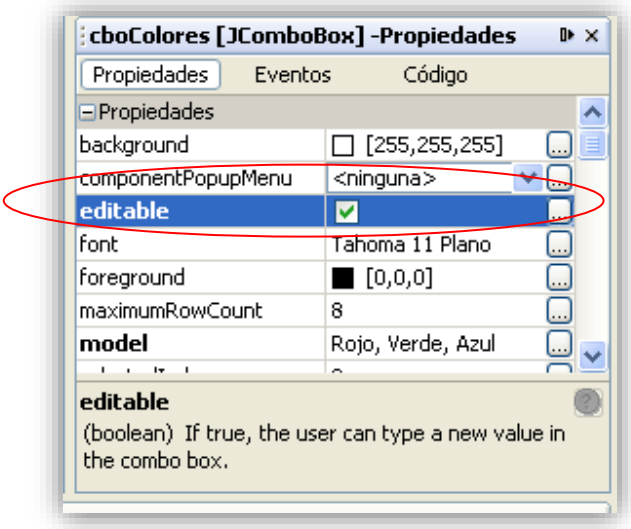
9. Este código hace lo siguiente:

    a. Crea una variable de cadena.
    b. Concatena dentro de ella el mensaje “El color elegido es” con el color seleccionado.
    c. Observa el método **getSelectedItem**, se usa para saber el elemento seleccionado del combo. Es necesario convertirlo a texto con toString.
    d. Finalmente se coloca el mensaje en la etiqueta.

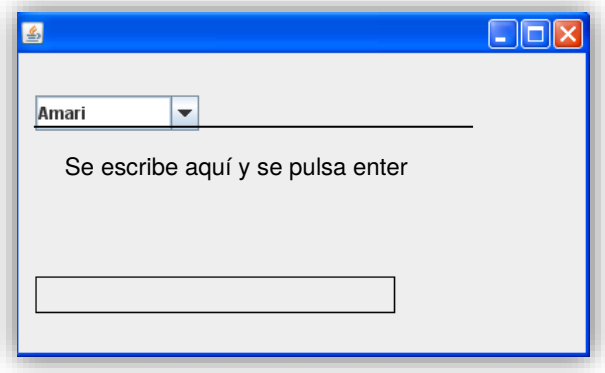
10. Ejecuta el programa y comprueba su funcionamiento. Por ejemplo, si elegimos el color verde, el aspecto del programa será el siguiente:



11. Los cuadros combinados pueden funcionar también como cuadros de texto. Es decir, pueden permitir que se escriba texto dentro de ellos. Para hacer esto, basta con cambiar su propiedad “editable” y activarla.



12. Ejecuta el programa y observa como se puede escribir dentro del combo. Al pulsar Enter, el programa funciona igualmente con el texto escrito.



CONCLUSIÓN

Los combos son listas desplegables donde se puede elegir una de las opciones propuestas.

Los combos pueden funcionar también como cuadros de textos, si se activa la opción editable.

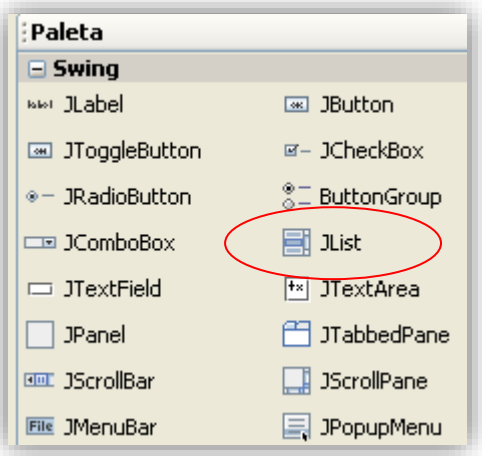
A través del método getSelectedItem se puede extraer la opción seleccionada o el texto escrito en el combo.

Ejercicio.3.S2.5. MODELOS DE CUADRO DE LISTA
Video en YouTube (U3S2 E05): https://www.youtube.com/watch?v=wz1XK7OFbB4

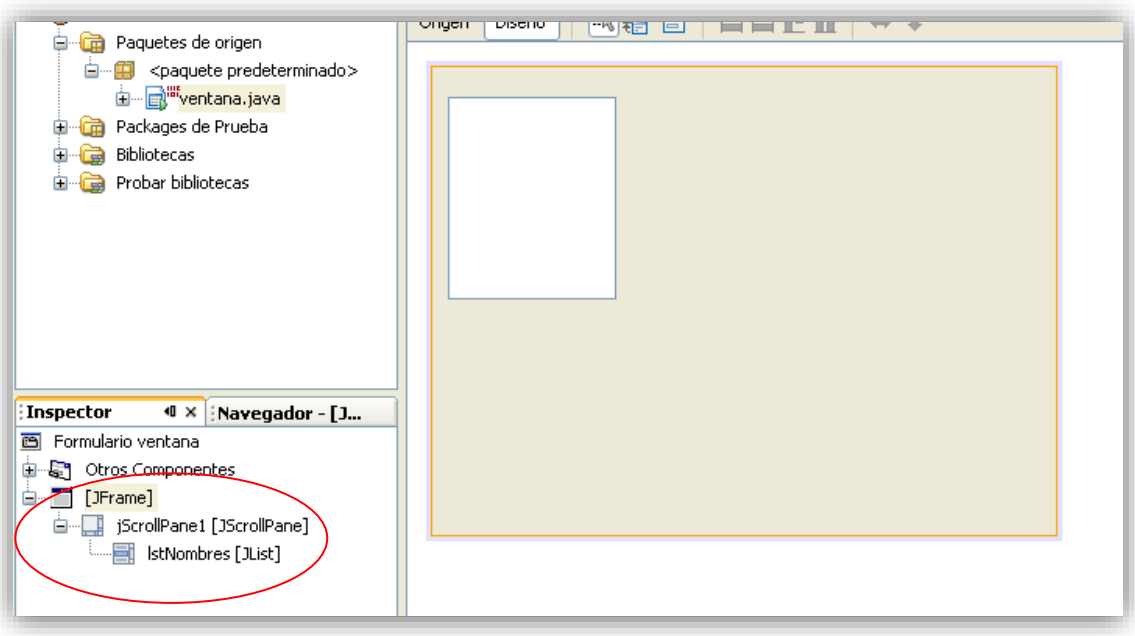
1. Realiza un nuevo proyecto.

Haz click en el link para verlo

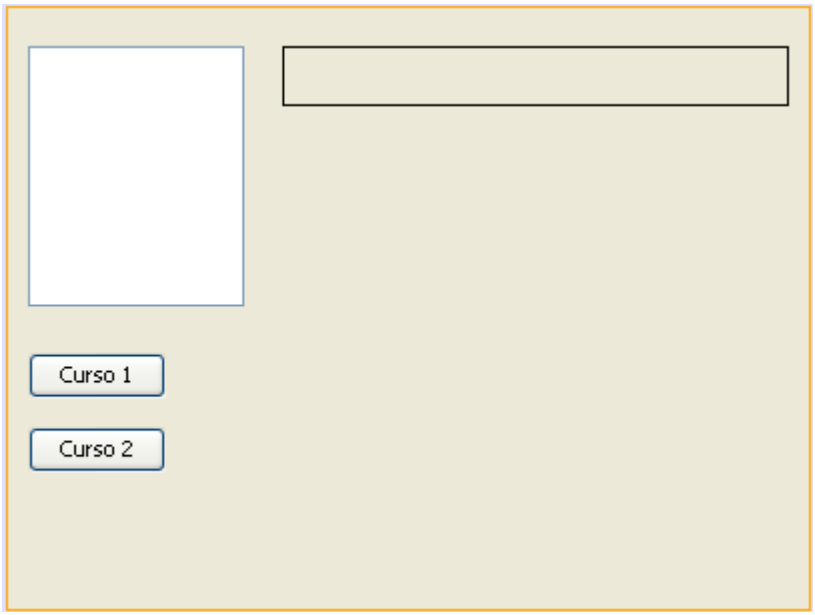
2. En la ventana principal debes añadir lo siguiente:
    a. Una etiqueta con borde llamada etiResultado.
3. Añade un cuadro de lista al formulario (JList).



4. Borra todo el contenido de la lista (propiedad model) y cámbiale el nombre a la lista. La lista se llamará *lstNombres*. Recuerda que las listas aparecen dentro de un objeto del tipo JScrollPane.



5. Añade dos botones al formulario. Uno de ellos tendrá el texto “Curso 1” y se llamará btnCurso1 y el otro tendrá el texto “Curso 2” y se llamará btnCurso2.



6. En el evento *actionPerformed* del botón “Curso 1” programa lo siguiente:

```
DefaultListModel modelo = new DefaultListModel();
modelo.addElement("Juan");
modelo.addElement("María");
modelo.addElement("Luis");
lstNombres.setModel(modelo);
```

7. En el evento *actionPerformed* del botón “Curso 2” programa lo siguiente:

```
DefaultListModel modelo = new DefaultListModel();
modelo.addElement("Ana");
modelo.addElement("Marta");
modelo.addElement("Jose");
lstNombres.setModel(modelo);
```

8. Explicación de los códigos anteriores:

    a. Lo que hace cada botón es rellenar el cuadro de lista con una serie de nombres. En el caso del botón “Curso 1”, la lista se rellena con los nombres Juan, María y Luis, mientras que en el caso del botón “Curso 2”, la lista se rellena con los nombres Ana, Marta y Jose.

    b. El contenido de un cuadro de lista es lo que se denomina un “modelo”. El “modelo” es un objeto que contiene el listado de elementos de la lista.

    c. Los modelos de las listas son objetos del tipo *DefaultListModel*.

    d. Lo que hace el programa es crear un “modelo”. Luego rellena el “modelo” con datos, y finalmente asocia el “modelo” al cuadro de lista. Veamos como se hace todo esto.

    e. Primero se crea el “modelo”, a través de la siguiente instrucción (será necesario añadir el *import* correspondiente, atento a la bombillita):

```
DefaultListModel modelo = new DefaultListModel();
```

    f. El “modelo” tiene un método llamado ***addElement*** que permite introducir datos dentro de él. Así pues usamos este método para añadir los datos al modelo.

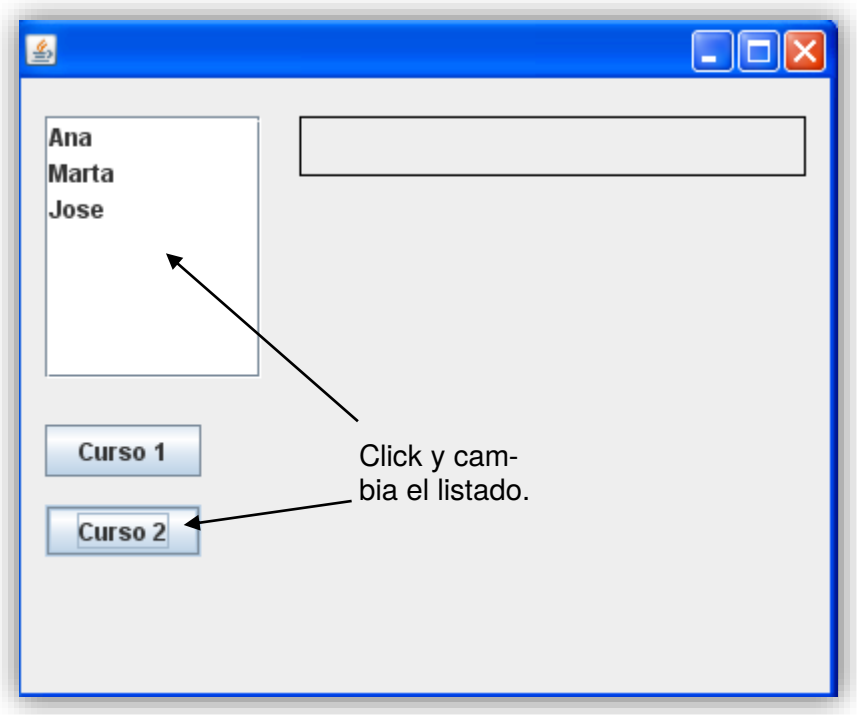
```
modelo.addElement("Ana");
modelo.addElement("Marta");
modelo.addElement("Jose");
```

    g. Finalmente asociamos el “modelo” creado al cuadro de lista de la siguiente forma:

```
lstNombres.setModel(modelo);
```

h. Así pues, aquí tienes una forma de cambiar el contenido de un cuadro de lista desde el propio programa.

9. Prueba a ejecutar el programa. Observa como cuando pulsas cada botón cambia el contenido de la lista:

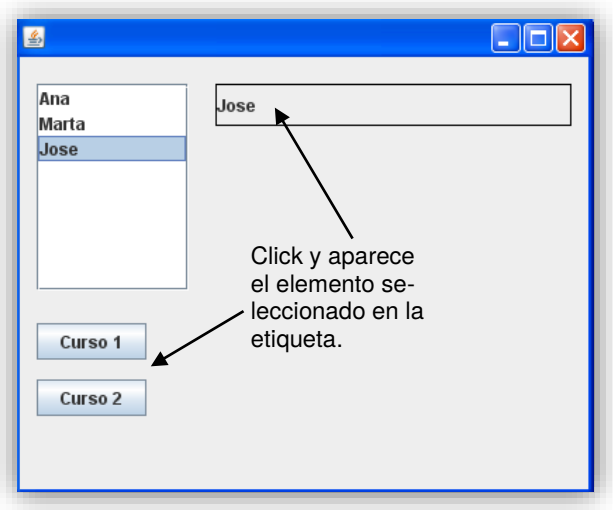


10. Ahora añade el siguiente código al evento *mouseClicked* del cuadro de lista:

```
etiResultado.setText(lstNombres.getSelectedValue().toString());
```

Esta instrucción hace que al seleccionar un elemento del cuadro de lista éste aparezca en la etiqueta etiResultado. Recuerda que el método getSelectedValue permite recoger el elemento seleccionado (hay que convertirlo a cadena con toString)

11. Ejecuta el programa:



12. Una propuesta. Añada un botón "Vaciar" llamado btnVaciar. Este botón vaciará el contenido de la lista. Para esto lo único que tiene que hacer es crear un modelo y, *sin introducir ningún valor en él*, asociarlo al cuadro de lista.

CONCLUSIÓN

Un cuadro de lista es un objeto que contiene a su vez otro objeto denominado "modelo".

El objeto "modelo" es el que realmente contiene los datos de la lista.

Cuadro de lista → Modelo → Datos

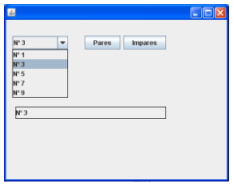
Se puede crear un "modelo" y luego introducir datos en él. Luego se puede asociar ese "modelo" a la lista. De esta manera se puede cambiar el contenido de la lista en cualquier momento.

Ejercicio.3.S2.6. MODELOS DE CUADRO DE LISTA
Video en You Tube (U3S2 E06): https://youtu.be/0wL77JlKEx4

Haz click en el link para verlo

1. Realiza un nuevo proyecto.

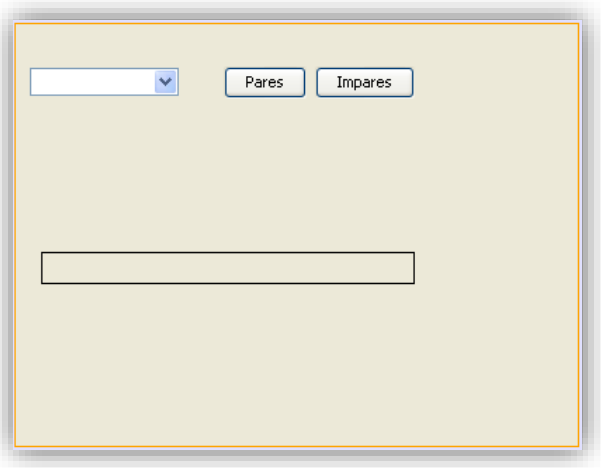
2. En la ventana principal debes añadir lo siguiente:



   a. Un combo llamado cboNumeros.
   b. Un botón "Pares" llamado btnPares.
   c. Un botón "Impares" llamado btnImpares.
   d. Una etiqueta con borde llamada etiResultado.

3. Elimina todos los elementos que contenga el combo. Recuerda, debes usar la **propiedad "model"** del combo para cambiar sus elementos.

4. Después de haber hecho todo esto, tu ventana debe quedar más o menos así:



5. En el evento *actionPerformed* del botón Pares, programa lo siguiente:

```
int i;

DefaultComboBoxModel modelo = new DefaultComboBoxModel();

for (i=0;i<10;i+=2) {
   modelo.addElement("N° "+i);
}

cboNumeros.setModel(modelo);
```

6. Observa lo que hace este código:
   a. Crea un objeto "modelo" para el combo.

      Al igual que pasa con los cuadros de lista, los combos tienen un objeto "modelo" que es el que realmente contiene los datos. En el caso de los combos, para crear un objeto "modelo" se usará esta instrucción:

```
DefaultComboBoxModel modelo = new DefaultComboBoxModel();
```

   b. A continuación, se usa el objeto "modelo" creado y se rellena de datos. Concretamente, se rellena con los números pares comprendidos entre 0 y 10.

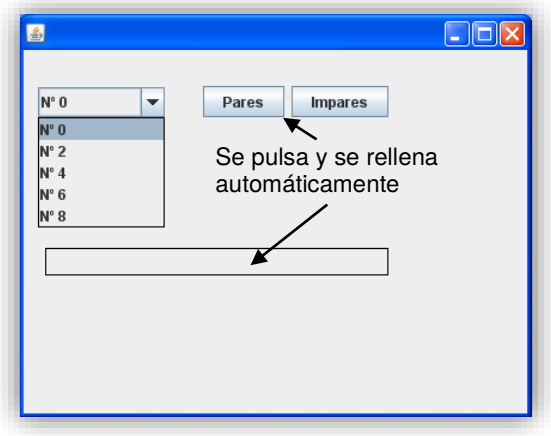
   c. Observa el uso de la propiedad addElement para añadir un elemento al modelo del combo.

   d. Se ha usado un bucle for para hacer la introducción de datos en el modelo más fácil.

e. Finalmente, se asocia el modelo al combo a través de la siguiente línea, con lo que el combo aparece relleno con los elementos del modelo:

```
cboNumeros.setModel(modelo);
```

7. Ejecuta el programa y observa el funcionamiento del botón Pares.



8. El botón Impares es similar. Programa su *actionPerformed* como sigue:

```
int i;
DefaultComboBoxModel modelo = new DefaultComboBoxModel();

for (i=1;i<10;i+=2) {
   modelo.addElement("N° "+i);
}

cboNumeros.setModel(modelo);
```

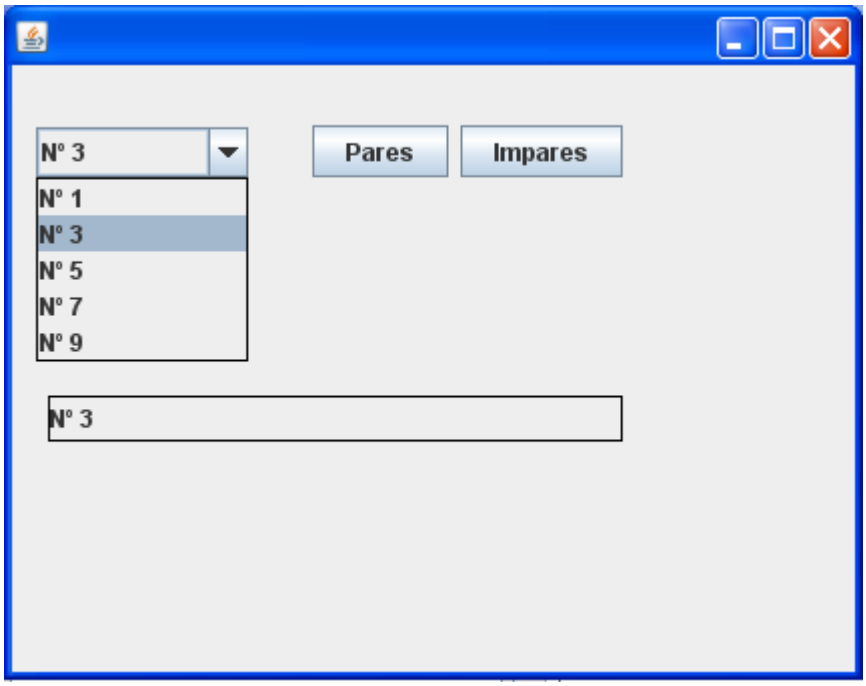
9. La única diferencia de este código es el for, que está diseñado para que se introduzcan los números impares comprendidos entre 0 y 10 dentro del modelo.

10. Finalmente se programará el *actionPerformed* del combo para que al seleccionar un elemento este aparezca en la etiqueta. Esto se hace con una simple instrucción:

```
etiResultado.setText(cboNumeros.getSelectedItem().toString());
```

Recuerda el uso de getSelectedItem() para recoger el elemento seleccionado, y el uso de toString() para convertirlo a texto.

11. Prueba el programa. Prueba los botones Pares e Impares y prueba el combo.



12. Sería interesante añadir un botón "Vaciar" llamado btnVaciar que vaciara el contenido del combo. Esto se haría simplemente creando un modelo vacío y asignarlo al combo. Se anima al alumno a que realice esta mejora.

CONCLUSIÓN

Un combo, al igual que los cuadros de lista, es un objeto que contiene a su vez otro objeto denominado "modelo".

El objeto "modelo" es el que realmente contiene los datos del combo.

Combo → Modelo → Datos

Se puede crear un "modelo" y luego introducir datos en él. Luego se puede asociar ese "modelo" al combo. De esta manera se puede cambiar el contenido del combo en cualquier momento.

Ejercicio.3.S2.7. TOGGLEBUTTONS

Video en You Tube (U3S2 E07) : https://youtu.be/puUENbzl7KI

1. Realiza un nuevo proyecto.
2. Crearás una ventana (Formulario JFrame...):

Ejercicio.3.S2.7. TOGGLEBUTTONS

3.

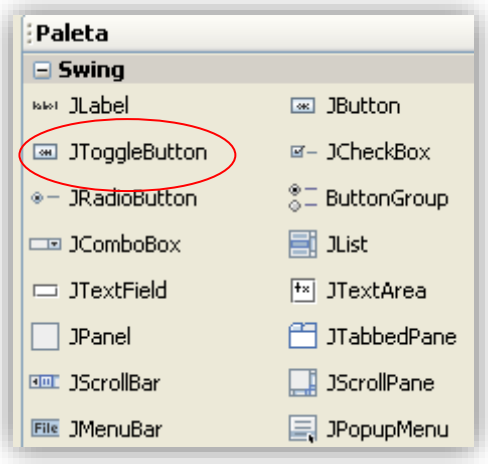
a. Se añadirá una **etiqueta** con el texto "Precio Base" en Bitcoins. No hace falta cambiarle su nombre.

b. Se añadirá un **cuadro de texto** llamado **txtPrecioBase**.

c. Se creará un **botón "Calcular"**, llamado **btnCalcular.**

d. Se creará una **etiqueta vacía y con borde** llamada **etiTotal**. Use la propiedad *font* de esta etiqueta para hacer que el texto tenga un mayor tamaño.

e. Debes añadir también tres **JToggleButton**, con el texto **"Instalación"**, **"Formación"** y **"Alimentación BD"** respectivamente.

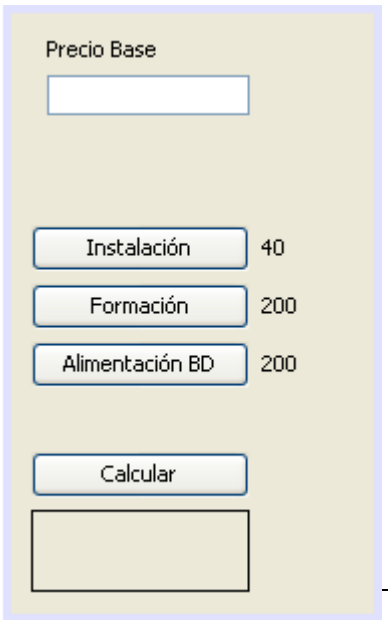


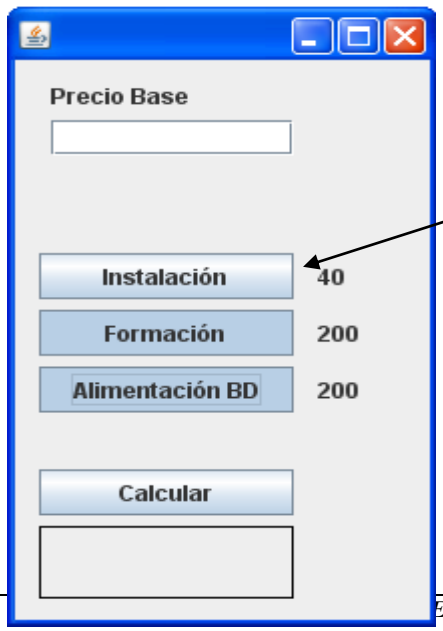
Estos botones no son botones normales, son botones del tipo **JToggleButton**. Usa este tipo de objeto para crearlos.
Estos botones, se diferencian de los botones normales en que se quedan pulsados cuando se hace un clic sobre ellos, y no vuelven a su estado normal hasta que no se vuelve a hacer clic sobre ellos.

Los tres botones se llamarán respectivamente: **tbtnInstalacion, tbtnFormacion, tbtnAlimentacionBD.**

f. Añade finalmente **tres etiquetas** conteniendo los números **40, 200, 200 Bitcoins.** La Primera se llamará **etiPrecioInstalacion**, la segunda **etiPrecioFormacion** y la tercera **etiPrecioAlimentacionBD**.

4. Prueba el programa y comprueba el funcionamiento de los botones JToggleButton:





Observa como al pulsar los JToggledButton estos se quedan pulsados.

Se vuelven a activar se *pulsan nuevamente*.

5. Se pretende que el programa funcione de la siguiente forma:

   a. El usuario introducirá un precio base para el servicio que se vende.
   b. A continuación, si el cliente quiere la instalación, activará el botón Instalación.
   c. Si el cliente quiere la formación, activará el botón Formación.
   d. Si el cliente quiere la Alimentación de Base de Datos, activará el botón Alimentación BD.
   e. Ten en cuenta que el cliente puede querer una o varias de las opciones indicadas.
   f. Finalmente se pulsará el botón calcular y se calculará el precio total. Este precio se calcula de la siguiente forma:

   Precio Total = Precio Base + Precio Extras.

   El precio de los Extras dependerá de las opciones elegidas por el usuario. Por ejemplo, si el usuario quiere Instalación y Formación, los extras costarán 240 Bitcoins.

6. Así pues, se programará el *actionPerformed* del botón Calcular de la siguiente forma:

```
double precio_base; //precio base
double precio_instal; //precio instalación
double precio_for; //precio formacion
double precio_ali; //precio alimentacion

//Recojo datos desde la ventana:

precio_base = Double.parseDouble(txtPrecioBase.getText());
precio_instal = Double.parseDouble(etiPrecioInstalacion.getText());
precio_for = Double.parseDouble(etiPrecioFormacion.getText());
precio_ali = Double.parseDouble(etiPrecioAlimentacionBD.getText());

//Ahora que tengo los datos, puedo hacer cálculos.

//Al precio base se le van añadiendo precio de extras
//según estén o no activados los JToggleButtons

double precio_total;

precio_total = precio_base;

if (tbtnInstalacion.isSelected()) { //Si se seleccionó instalación
   precio_total = precio_total+precio_instal;
}

if (tbtnFormacion.isSelected()) { //Si se seleccionó formación
   precio_total = precio_total+precio_for;
}

if (tbtnAlimentacionBD.isSelected()) { //Si se seleccionó Alimentación BD
   precio_total = precio_total+precio_ali;
}

//Finalmente pongo el resultado en la etiqueta
etiTotal.setText(precio_total+" $BC");
```

7. Veamos una explicación del código:

   a. Primero se crean variables doubles (ya que se admitirán decimales) para poder hacer los cálculos.

   b. Se extraerán los datos de la ventana y se almacenarán en dichas variables. Para ello, hay que convertir desde cadena a double:

```
precio_base = Double.parseDouble(txtPrecioBase.getText());
precio_instal = Double.parseDouble(etiPrecioInstalacion.getText());
precio_for = Double.parseDouble(etiPrecioFormacion.getText());
precio_ali = Double.parseDouble(etiPrecioAlimentacionBD.getText());
```

   c. Una vez obtenidos los datos en forma numérica, ya se pueden hacer cálculos con ellos. El programa declara una nueva variable *precio_total* donde se introducirá el resultado. En primer lugar se introduce en esta variable el precio base.

```
double precio_total;

precio_total = precio_base;
```

   d. A continuación se le suma al precio_total los precios de los extras si el botón correspondiente está seleccionado. Esto se hace a través de **"if"**. La estructura condicionar **"if"** permite tomar decisiones en un algoritmo o en un programa. La sintaxis que deberemos emplear en Java es la siguiente:

```
if <expresión booleana>
  { sentencias 1, por verdadero }
  else {sentencias 2, por falso };
```

   Esta sentencia indica al compilador que la expresión boolean que colocamos después del **If** arroja un valor verdadero (True), se debe ejecutar el bloque de sentencias 1; en caso contrario ejecutará el bloque de sentencias 2 .

   Por ejemplo, para sumar el extra por instalación:

```
if (tbtnInstalacion.isSelected()) { //Si se seleccionó instalación
   precio_total = precio_total+precio_instal;
}
```

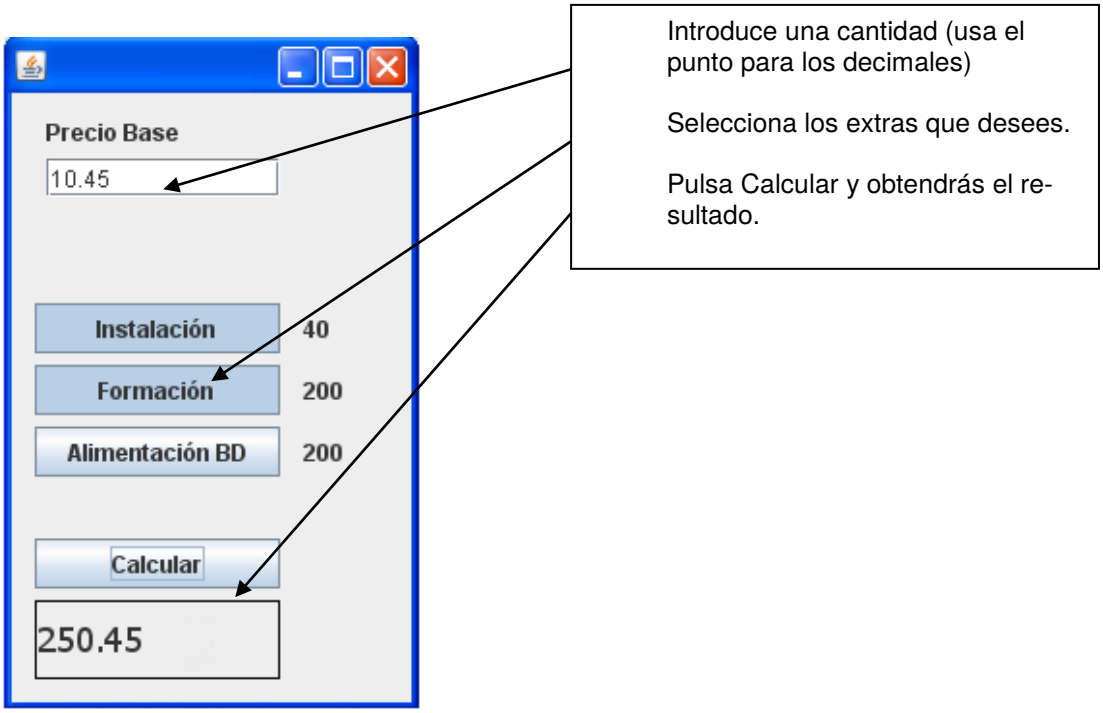
   Esto significa: "Si el botón instalación está seleccionado, añade al precio total el precio por instalación"

```
tbtnInstalacion.isSelected()- "Devolvera true o false"
```

   Observa el uso del método ***isSelected*** para saber si el botón tbtnInstalacion ha sido seleccionado.

   e. Finalmente el resultado se muestra en la etiqueta de total.

8. Comprueba el funcionamiento del programa…



9. Supongamos que normalmente (en el 90 por ciento de los casos) la instalación es solicitada por el usuario. Podría ser interesante que el botón Instalación ya saliera activado al ejecutarse el programa. Para ello, añade en el *Constructor* la siguiente línea.

```
tbtnInstalacion.setSelected(true);
```

Esta línea usa el método ***setSelected*** para activar al botón **tbtnInstalación.**

Comprueba esto ejecutando el programa.

CONCLUSIÓN

Los JToggleButton son botones que pueden quedarse pulsados.

A través del método isSelected podemos saber si un JToggleButton está seleccionado.

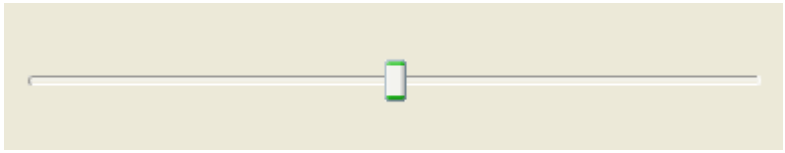
También puedes usar el método setSelected para seleccionar o no un botón de este tipo.

Realmente, estos botones no suelen ser muy usados, ya que pueden ser sustituidos por Cuadros de Verificación (JCheckBox) que son más conocidos.

Ejercicio.3.S2.8. SLIDERS

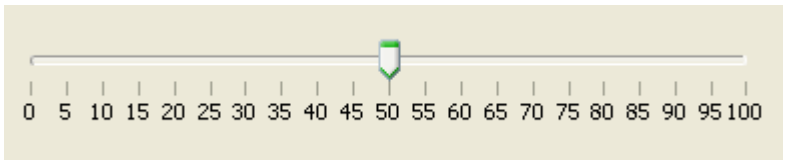
Introducción a los JSliders

La clase javax.swing.**JSlider** permite crear objetos como el siguiente:

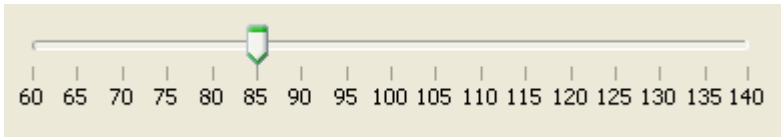


Estos elementos tienen un pequeño recuadro que se puede arrastrar a derecha o izquierda. Según la posición del recuadro, el **JSlider** tendrá un valor concreto.

El **JSlider** se puede configurar para que muestre los distintos valores que puede tomar:



También se puede configurar de forma que los valores mínimo y máximo sean distintos:



El valor que tiene un **JSlider** es el valor al que apunta el recuadro del **JSlider**. En la imagen anterior, el **JSlider** tiene un valor de 85.

Se verá a continuación las características más interesantes de los **JSlider** y como programarlos.

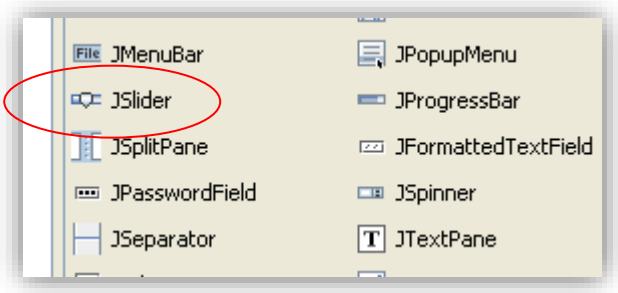
Ejercicio guiado paso a paso

Video en You Tube (U3S2 E08) : https://youtu.be/-UOxFsibhMY

Swing Control - Componente SLider

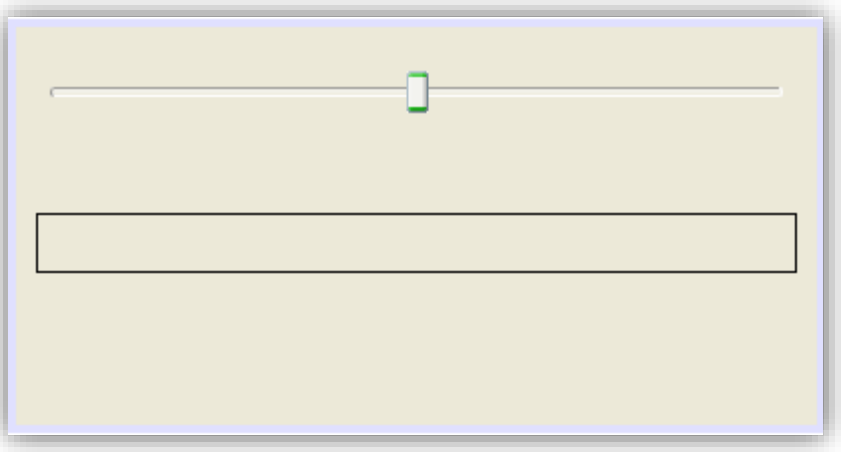
1. Realiza un nuevo proyecto.
2. Crearás una ventana (Formulario JFrame...) como la que sigue tenien en cuenta lo siguiente:

3. Añade en él un **JSLider**. Su nombre será `slDeslizador`.



4. Añade una etiqueta con borde. Su nombre será *etiValor*.

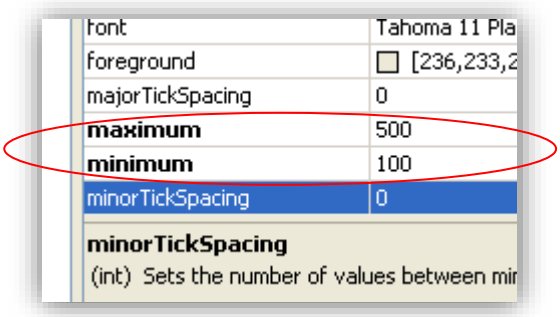
5. La ventana tendrá el siguiente aspecto:



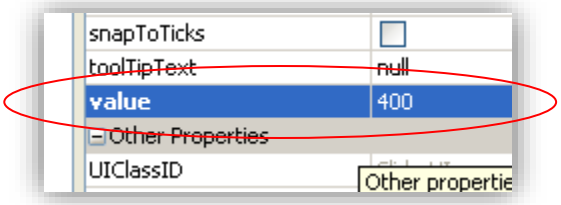
6. Un **JSlider** tiene un valor mínimo y un valor máximo. El valor mínimo es el valor que tiene cuando el recuadro está pegado a la parte izquierda, mientras que el valor máximo es el valor que tiene cuando el recuadro está pegado a la parte derecha.

   El valor mínimo y máximo del JSlider se puede cambiar. Busca las propiedades ***maximum*** **y** ***minimum*** del **JSlider** y asigna los siguientes valores:

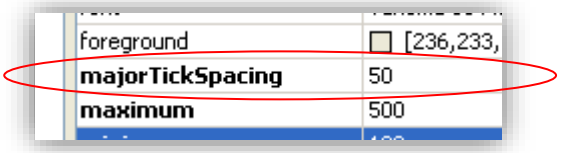
   Máximo: 500
   Mínimo: 100



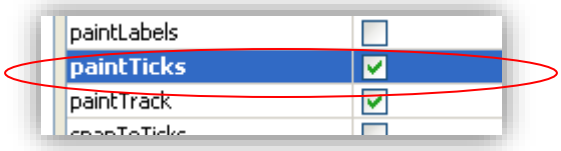
7. Se puede asignar un valor inicial al **JSlider** a través de su propiedad ***value***. Busque esta propiedad y asigne un valor de 400. Observe donde se sitúa el recuadro del **JSlider**.

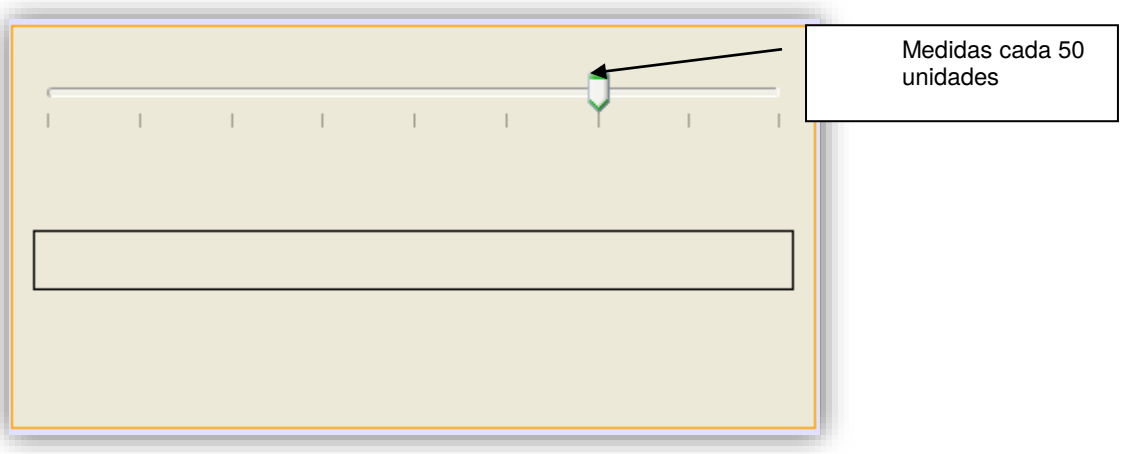


8. Se puede mejorar el **JSlider** definiendo unas divisiones (medidas) Por ejemplo, haremos que cada 50 unidades aparezca una división. Para ello use la propiedad ***majorTickSpacing*** y asigne un 50.

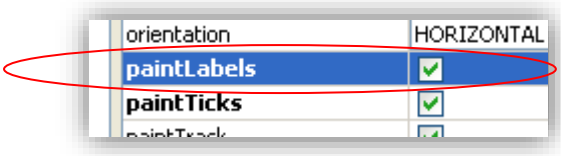


9. Esto, en realidad, no produce ningún cambio en el **JSlider**. Para que las divisiones se vean, es necesario que active también la propiedad ***paintTicks***. Esta propiedad pintará divisiones en el **JSlider**:

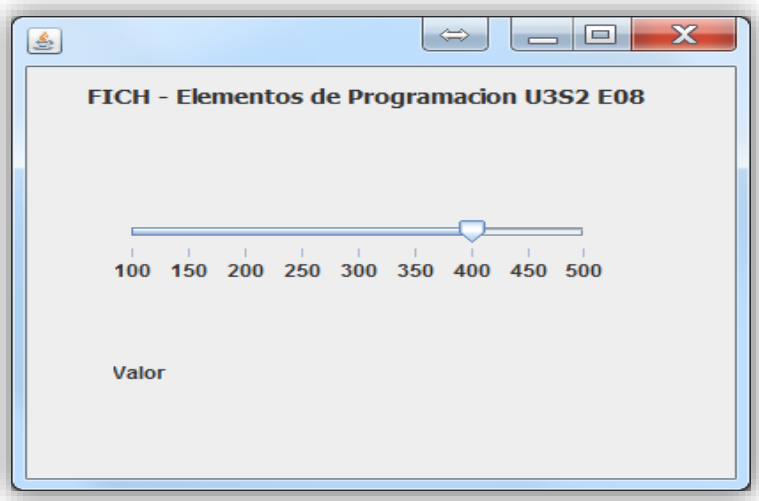




10. Aún se puede mejorar la presentación del **JSlider**, si hacemos que aparezca el valor de cada división.
    Para ello debes activar la propiedad ***paintLabel***.



11. Ejecuta el programa para ver el funcionamiento del Deslizador y su aspecto. Debe ser parecido al siguiente:



12. Bien. Ahora se pretende que cuando el usuario arrastre el deslizador, en la etiqueta aparezca el valor correspondiente. Para ello tendrá que programar el evento ***stateChanged*** del **JSlider**.

    El evento ***stateChanged*** sucede cuando el usuario arrastra el recuadro del deslizador.

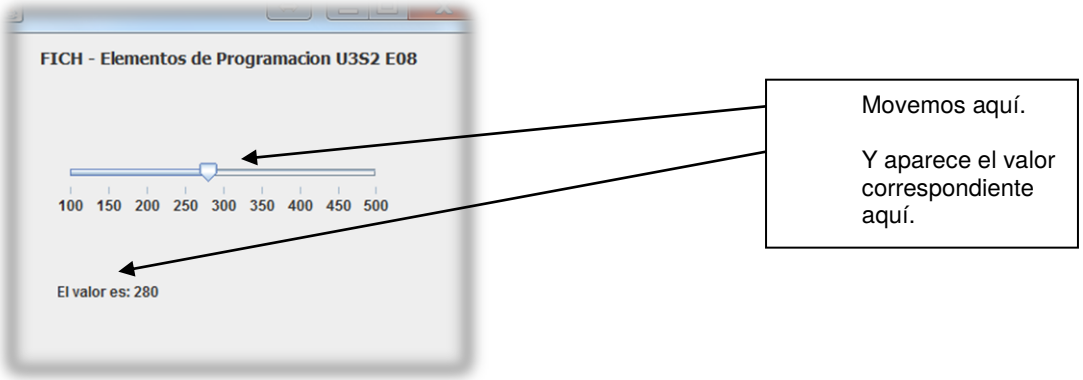
En este evento programe lo siguiente:

```
etiValor.setText("El valor es:
"+slDeslizador.getValue());
```

13. Ejecute el programa y observe lo que sucede cuando arrastra el deslizador.

14. La explicación del código es la siguiente:

    a. El método *getValue* del deslizador nos devuelve el valor que tiene actualmente el deslizador.
    b. Este valor es concatenado a la cadena "El valor es:" y es mostrado en la etiqueta a través del conocido *setText*.



15. A continuación se mencionan otras propiedades interesantes de los JSlider que puedes probar por tu cuenta:

orientation
Permite cambiar la orientación del JSlider. Podrías por ejemplo hacer que el JSlider estuviera en vertical.

minorTickSpacing
Permite asignar subdivisiones a las divisiones ya asignadas. Prueba por ejemplo a asignar un 10 a esta propiedad y ejecuta el programa. Observa las divisiones del JSlider.

snapToTicks
Cuando esta propiedad está activada, no podrás colocar el deslizador entre dos divisiones. Es decir, el deslizador siempre estará situado sobre una de las divisiones. Prueba a activarla.

paintTrack
Esta propiedad permite pintar o no la línea sobre la que se desliza el JSlider. Prueba a desactivarla.

CONCLUSIÓN

Los JSliders son objetos "deslizadores". Permiten elegir un valor arrastrando un pequeño recuadro de derecha a izquierda o viceversa.

El valor de un JSliders puede ser obtenido a través de su método *getValue*.

Si quieres programar el cambio (el arrastre) en el deslizador, tienes que programar el evento llamado *stateChanged*.

Ejercicio3.S2.9. SPINNER

Video en You Tube (U3S2 E09) : https://youtu.be/dN-93MxNaMs

Haz click en el link para verlo

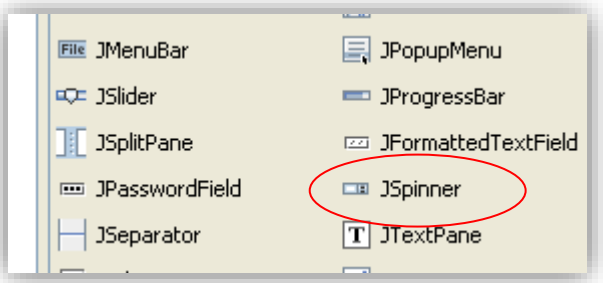
Introducción a los Spinner

La clase javax.swing.**JSpinner** permite crear cuadros como el siguiente:

Son elementos muy comunes en los programas. A través de los dos botones triangulares se puede hacer que el valor del cuadro aumente o disminuya. También se puede escribir directamente un valor dentro del cuadro.

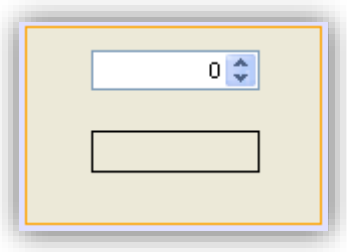
Ejercicio guiado paso a paso

1. Crea un nuevo proyecto.
2. Añade en él un JSpinner. Su nombre será `spiValor`.



3. Añade una etiqueta con borde. Su nombre será *etiValor*.

4. La ventana tendrá el siguiente aspecto:



5. Interesa que cuando cambie el JSpinner (ya sea porque se pulsaron los botones triangulares o porque se escribió dentro) aparezca el valor correspondiente dentro de la etiqueta. Para ello, tendrá que programar el evento *stateChanged* del JSpinner.

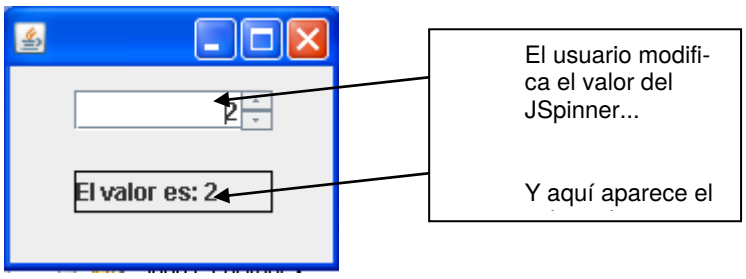
   En el evento *stateChanged* introduzca el siguiente código:

```
etiValor.setText("El valor es: "+spiValor.getValue().toString());
```

6. Como puedes observar, lo que hace el programa es recoger el valor que tiene el JSpinner a través del método *getValue* y luego se lo asigna a la etiqueta con el clásico *setText*. (Es muy parecido a los deslizadores)

   Debes tener en cuenta que el valor devuelto no es un número ni una cadena, así que en el ejemplo se ha usado el método *toString()* para convertirlo a una cadena.

7. Prueba el programa y observa su funcionamiento:



8. Observa como los valores del JSpinner aumentan o disminuyen en 1. Por otro lado, no parece haber un límite para los valores del JSpinner.

   La pregunta ahora es: ¿Se puede modificar el contenido del JSpinner de forma que tenga unos valores concretos? La respuesta es sí. Veamos como hacerlo.

9. Entra dentro del código del programa, pestaña Source, en el JFrame Form, dentro del constructor, al inicio del código, debajo de ***initComponents,*** añade este código:

```
SpinnerNumberModel nm = new SpinnerNumberModel();
nm.setMaximum(10);
nm.setMinimum(0);
spiValor.setModel(nm);
```

10. Este código hace lo siguiente:

    a. El JSpinner, al igual que los JList y los JComboBox, es un objeto que contiene otro objeto "modelo", y es el objeto "modelo" el que contiene los números visualizados en el JSpinner.

    b. En el código anterior se crea un "modelo" para el JSpinner, se definen los valores que contendrá, y luego se asigna al JSpinner. Estudiemos las líneas del código.

    c. La Segunda línea crea un "modelo" llamado *nm*. Los modelos de los JSpinner son del tipo *SpinnerNumberModel*. Necesitarás incluir el import correspondiente (atento a la bombilla)

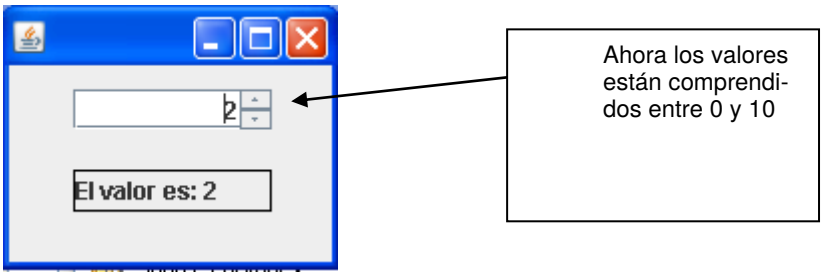
    d. En la segunda línea se define como valor máximo del modelo el 10, a través de un método llamado *setMaximum*.

    e. En la tercera línea se define como valor mínimo del modelo el 0, a través de un método llamado *setMinimum*.

    f. Finalmente se asigna el modelo creado al JSpinner.

    g. Este código, en definitiva, hará que el JSpinner muestre los valores comprendidos entre 0 y 10.

11. Prueba el programa y observa los valores que puede tomar el JSpinner.



12. Vamos a añadir otra mejora. Cambie el código del constructor por este otro. (Observa que solo se ha añadido una línea):

```
SpinnerNumberModel nm = new SpinnerNumberModel();
nm.setMaximum(10);
nm.setMinimum(0);
nm.setStepSize(2);
spiValor.setModel(nm);
```

13. La línea añadida es:

```
nm.setStepSize(2);
```

Esta línea usa un método del modelo del JSpinner que permite definir el valor de cambio del JSPinner. Dicho de otra forma, esta línea hace que los valores del JSpinner salten de 2 en 2.

14. Ejecuta el programa de nuevo y observa como cambian los valores del JSpinner.

15. El modelo del JSpinner tiene también un método llamado *setValue* que permite asignar un valor inicial al modelo. Pruebe a usar este método para hacer que el JSpinner muestre desde el principio el valor 4.

CONCLUSIÓN

Los JSpinners son objetos que permiten seleccionar un número, ya sea escribiéndolo en el recuadro, o bien a través de dos botones triangulares que permiten aumentar o disminuir el valor actual.

Los JSpinners son objetos con "modelo". Es decir, este objeto contiene a su vez otro objeto "modelo" que es el que realmente contiene los datos.

Datos → Modelo → JSpinner

Para definir el contenido del JSpinner es necesario crear un modelo del tipo SpinnerNumberModel. Se le asigna al modelo los números deseados, y finalmente se une el modelo con el JSpinner.

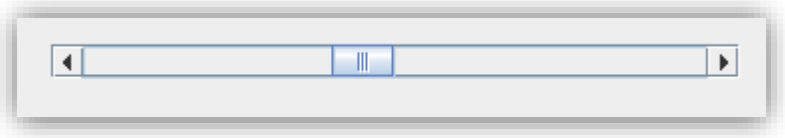
El objeto modelo del JSpinner permite definir el valor mínimo y el valor máximo, así como el intervalo de aumento de los valores.

Ejercicio.3.S2.10. SCROLLBARS

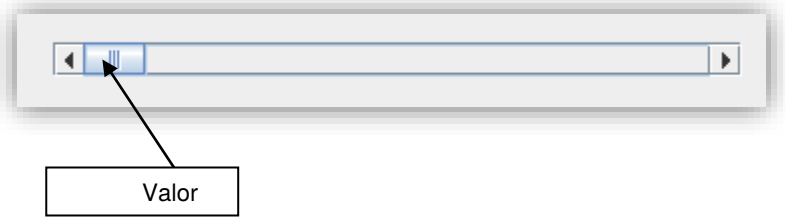
Video en You Tube (U3S2 E10) :

Introducción a las JscrollBars (Barras de desplazamiento)

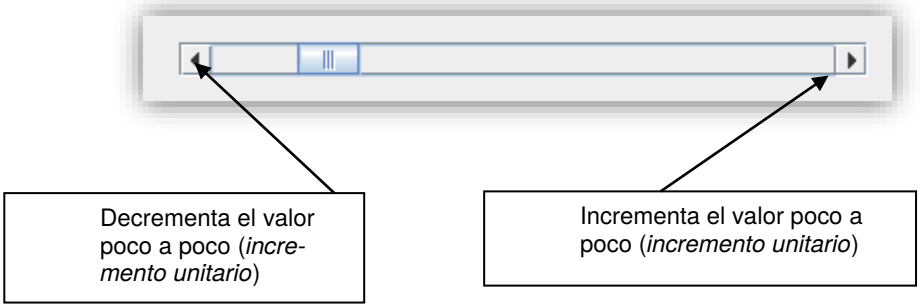
La clase **JScrollBar** permite crear barras de desplazamiento independientes, como la que se muestra a continuación:



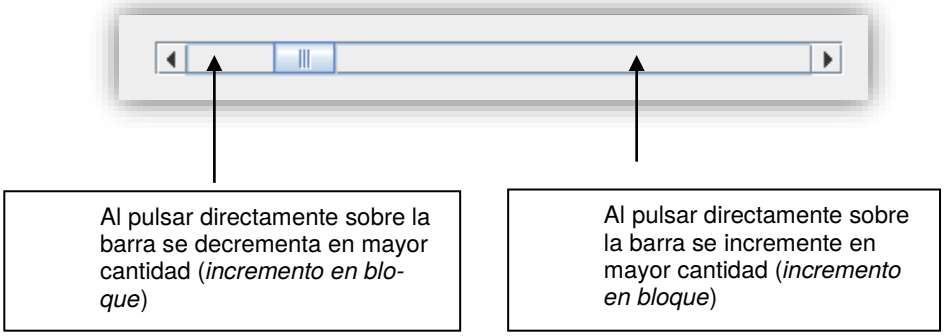
La barra tiene un valor mínimo, que se consigue haciendo que el recuadro de la barra de desplazamiento esté pegado a la parte izquierda.



Cuando se pulsa algunos de los botones de la barra de desplazamiento, el valor de la barra se incrementa / decrementa poco a poco. A este incremento / decremento lo llamaremos *incremento unitario*.



Cuando se pulsa directamente sobre la barra, el valor de la barra se incrementa / decrementa en mayor cantidad. A este incremento / decremento lo llamaremos *incremento en bloque*.



Ejercicio guiado paso a paso

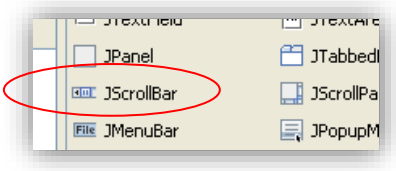
Para comprender mejor el funcionamiento de las barras de desplazamiento se creará un proyecto nuevo.

Video en You Tube (U3S2 E10) : https://youtu.be/DuYTgK9PZfc

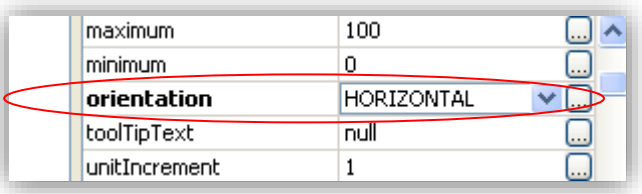
Haz click en el link para verlo

1. Realiza un nuevo proyecto.
2. Crearás una ventana (Formulario JFrame...) como la que sigue teniendo en cuenta lo siguiente:

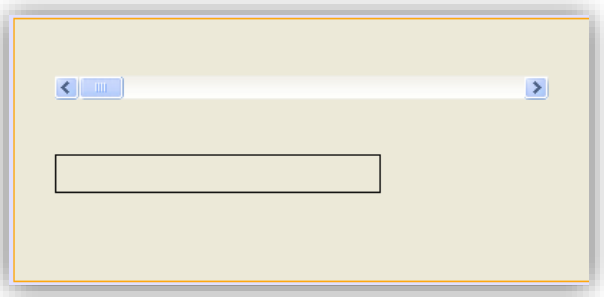
1. Añade en el proyecto una barra de desplazamiento (JScrollBar) y llámala ***desValor*.**



2. La barra de desplazamiento aparecerá en vertical. Use la **propiedad** de la barra llamada *Orientation* para hacer que la barra aparezca en posición horizontal.



3. Añade también una etiqueta con borde y llámala **etiValor**.
4. La ventana debe quedar más o menos así:



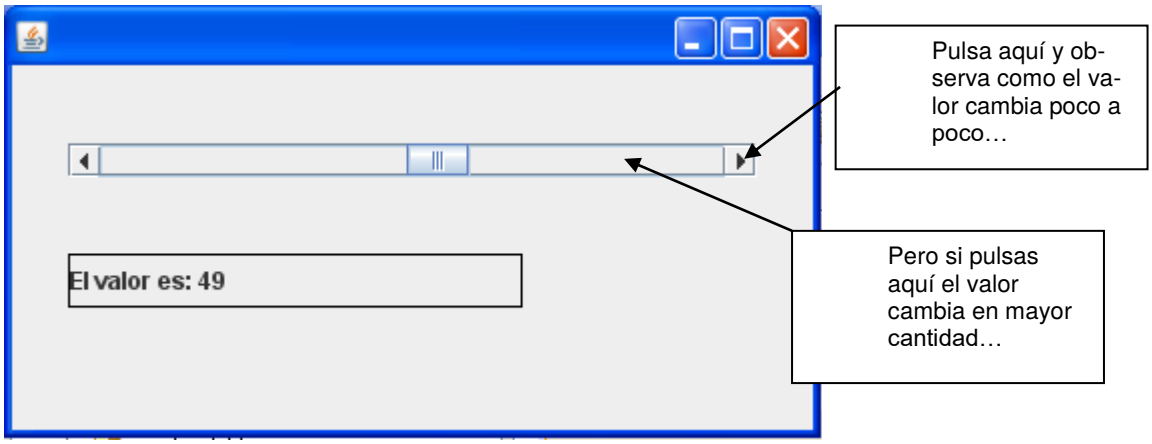
5. Interesa que cuando el usuario cambie de alguna manera la barra de desplazamiento, en la etiqueta aparezca el valor de la barra.

   Para ello, se debe programar el evento ***AdjustmentValueChanged*** de la barra de desplazamiento.

   En este evento programa lo siguiente:

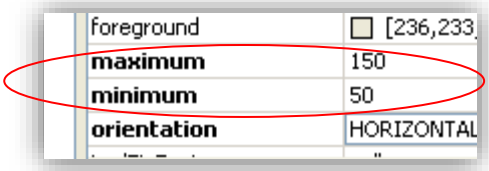
```
etiValor.setText("El valor es: "+desValor.getValue());
```

6. Como ves, se coloca en la etiqueta el valor de la barra. El valor de la barra se obtiene con el método *getValue*. Ejecuta el programa para ver su funcionamiento.

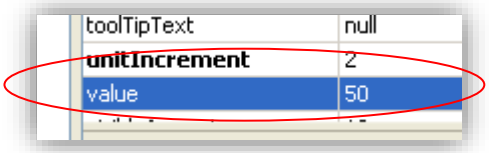


7. Sigamos estudiando el programa:
   Se pide que cambies las siguientes **propiedades** de tu barra:

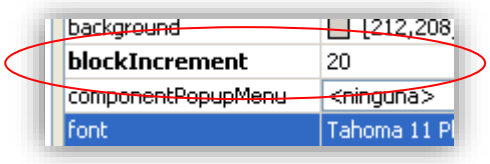
   Minimum – Permite asignar el valor mínimo de la barra. Escribe un 50
   Maximum – Permite asignar el valor máximo de la barra. Escribe un 150



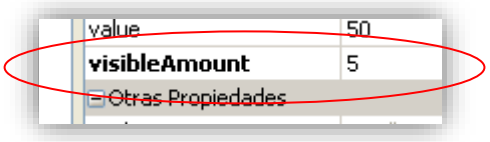
   UnitIncrement – Permite cambiar el *incremento unitario*. Escribe un 2.



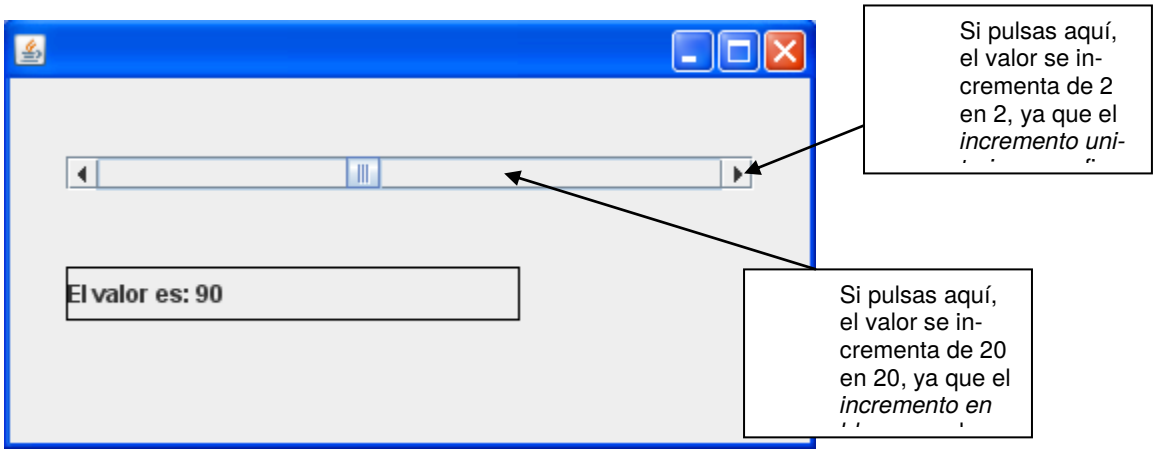
BlockIncrement – Permite cambiar el *incremento en bloque*. Escribe un 20.

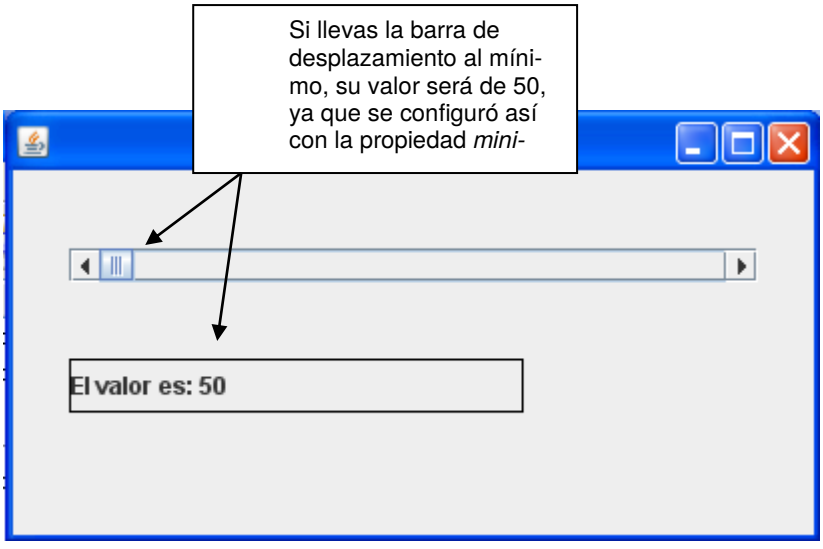


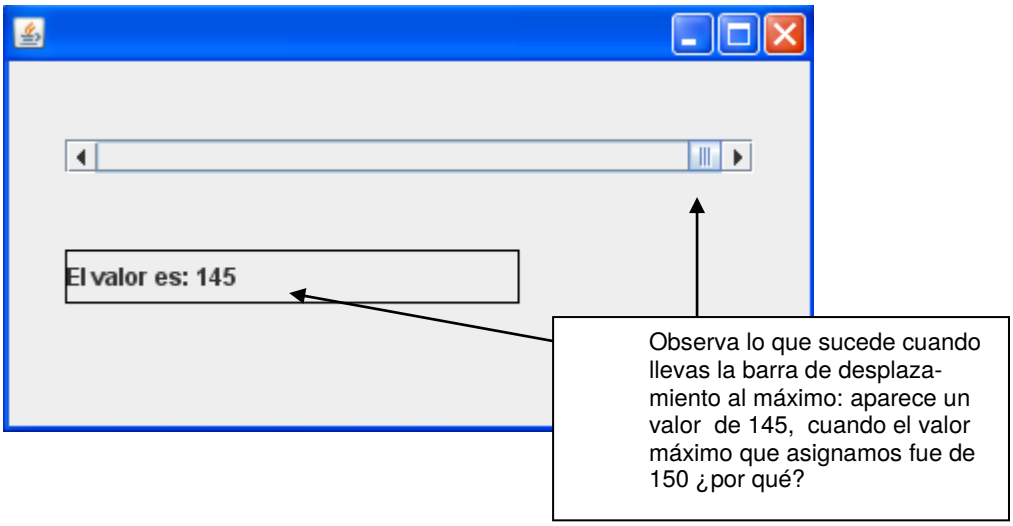
VisibleAmount – Permite cambiar el ancho del recuadro de la barra. Escribe un 5.



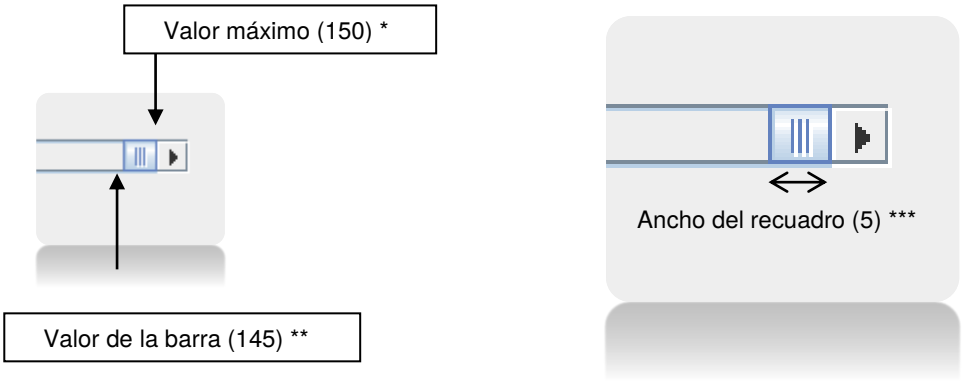
8. Ejecuta ahora el programa y comprueba su funcionamiento:







9. Tal como se ha indicado anteriormente, pasa algo raro con la barra de desplazamiento cuando esta está al máximo. Se esperaba que alcanzara el valor 150, y sin embargo, el valor máximo alcanzado fue de 145. La explicación es la siguiente:



* Nuestra barra tiene un valor máximo de 150.

** Sin embargo, el valor de la barra viene indicado por el lado izquierdo del recuadro interno.

*** Como el recuadro interno tiene un ancho definido a través de la propiedad *VisibleAmount*, el valor máximo que la barra puede alcanzar es de:

Valor = ValorMáximo – Ancho del recuadro.

Es decir,

Valor alcanzable = 150 – 5 = 145

10. A través del método *setValue* de la barra de desplazamiento se puede asignar un valor inicial a la barra. Programe en el constructor de su programa lo necesario para que la barra de desplazamiento tenga un valor de 70 al empezar el programa.

CONCLUSIÓN

Las JScrollBars son barras de desplazamiento independientes. Al igual que los JSliders, las JScrollBars tienen un valor concreto, que puede ser obtenido a través del método *getValue*.

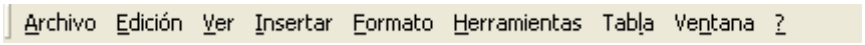
Entre las características programables de una barra de desplazamiento, tenemos las siguientes:

- **Valor mínimo (propiedad Minimum)**
- **Valor máximo (propiedad Maximum)**
- **Incremento unitario (propiedad UnitIncrement)**
- **Incremento en bloque (propiedad BlockIncrement)**
- **Tamaño del recuadro de la barra (propiedad VisibleAmount)**

Ejercicio.3.S2.11 BARRA DE MENUS - https://youtu.be/iBnQNEf4DDA

Barras de Menús

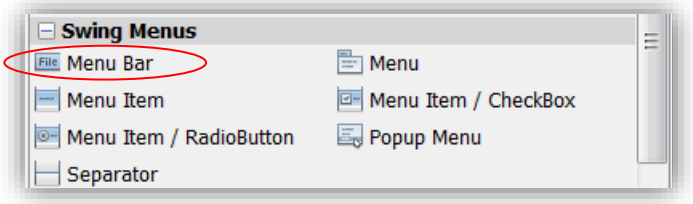
La barra de menús nos permitirá acceder a las opciones más importantes del programa. Todo programa de gran envergadura suele tener una barra de menús.



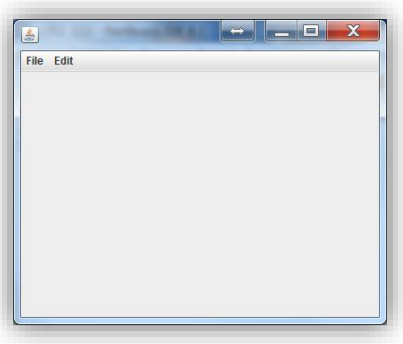
Ejercicio guiado paso a paso

Veamos como añadir una barra de menús a nuestras aplicaciones.

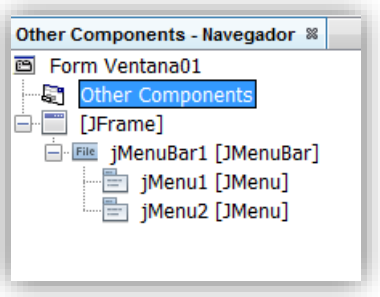
1. Realiza un nuevo proyecto.
2. Crearás una ventana (Formulario JFrame...) como la que sigue teniendo en cuenta lo siguiente:
3. Añade a tu ventana un objeto Swing Menus – Menu Bar (JMenuBar)



4. En la parte superior de tu ventana aparecerá esto:

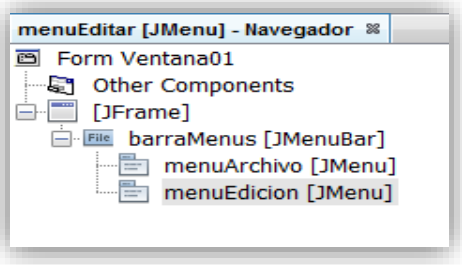


5. En el inspector (parte inferior izquierda) observarás como aparece un objeto JMenuBar, y, dentro de él, dos objetos del tipo JMenu. Los objetos JMenu representan las opciones principales contenidas dentro de la barra de menús. (JMenu1 y JMenu2)

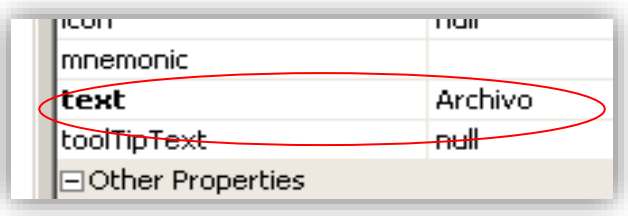


6. Aprovecha el *Inspector* para cambiar el nombre al objeto JMenuBar. Llámalo *barraMenus*.

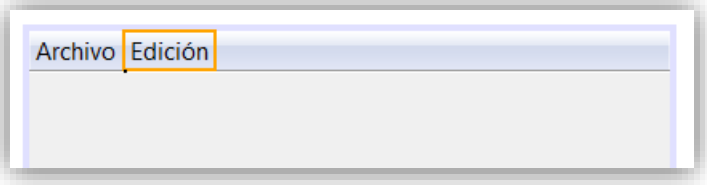
7. Cambia también el nombre al objeto jMenu1. Asígnale el nombre *menuArchivo* y a jMenu2, el nombre menuEdicion El *Inspector* tendrá el siguiente aspecto:



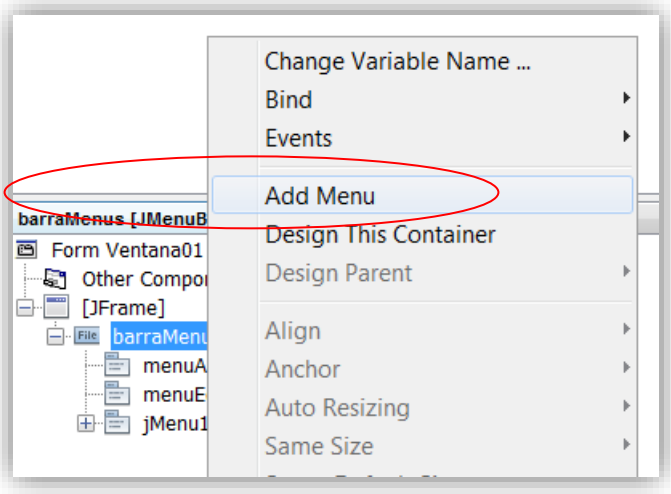
8. Ahora, la barra de menús muestra el texto “File” y “Edit”. Esto se puede cambiar seleccionándola y cambiando su propiedad *text*. Asígnale el texto “Archivo” y “Edición” a las opción del menú respectivamente:



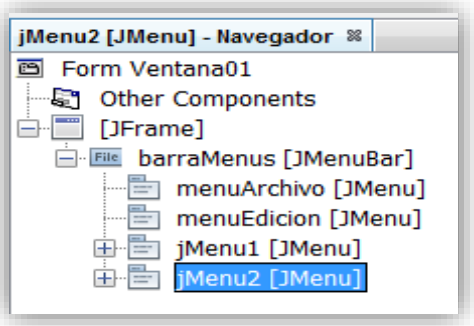
9. Ahora el aspecto de la barra de menús será el siguiente:



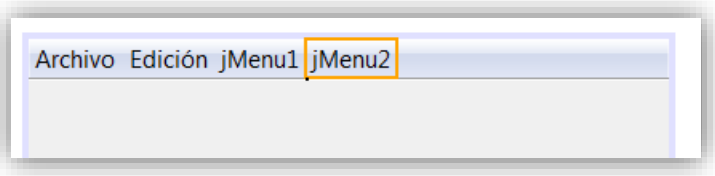
10. Puedes añadir más opciones principales a la barra de menús haciendo clic con el derecho sobre el objeto de la barra de menús y activando la opción “Add Menu”.



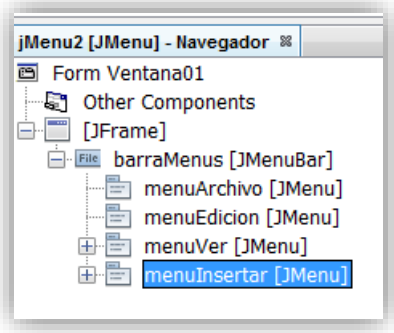
11. Añada dos opciones más a la barra de menús. El inspector debe tener ahora el siguiente aspecto:



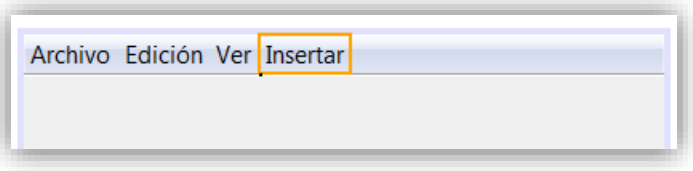
12. Y la barra de menús presentará este otro aspecto:



13. Cambia los nombres de las dos nuevas opciones. Sus nombres serán: menuVer y menuInsertar.

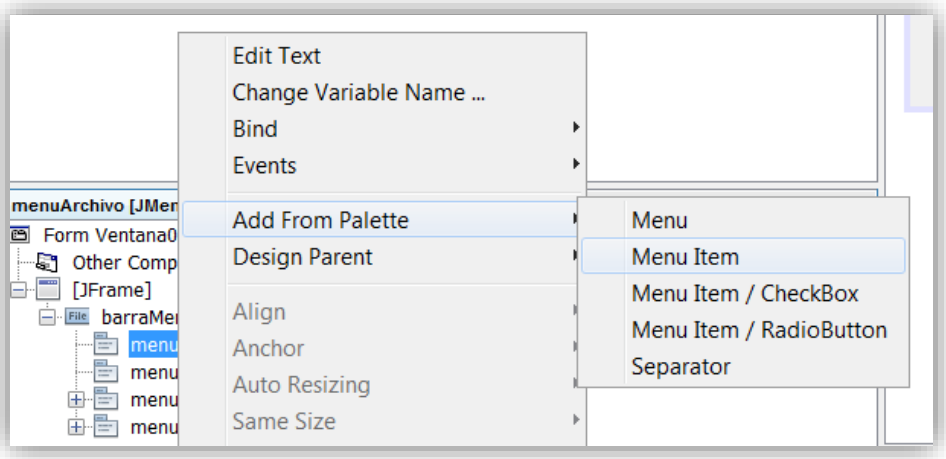


14. Cambia los textos de ambas opciones. Sus textos serán: “Ver” e “Insertar”.



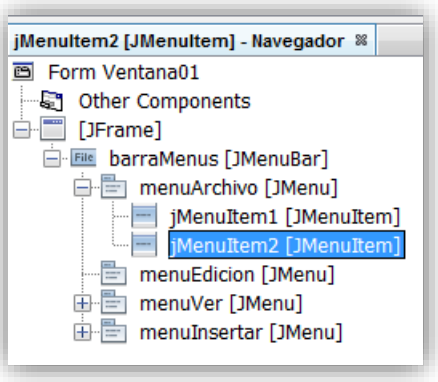
15. Ya tenemos creada la barra de menús (JMenuBar) con sus opciones principales (JMenu). Ahora se tendrán que definir las opciones contenidas en cada opción principal. Por ejemplo, crearemos las opciones contenidas en el menú Archivo.

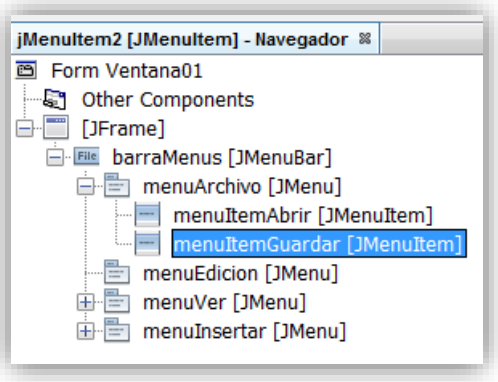
16. Haz clic con el botón derecho sobre el objeto menuArchivo y activa la opción Add From Palette - “Menu Item”.



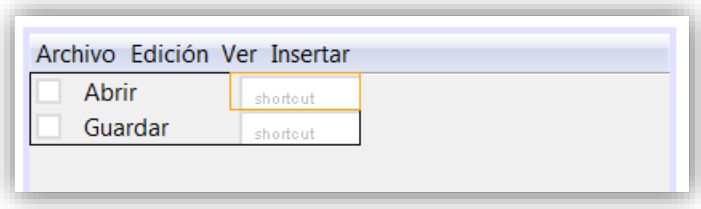
Los Menú Item son objetos que representan las opciones contenidas en los menús desplegables de la barra de menús.

17. Añade un Menú Item más al menuArchivo y luego cambia el nombre a ambos. Sus nombres serán *menuItemAbrir* y *menuItemGuardar*. El aspecto del *Inspector* será el siguiente:



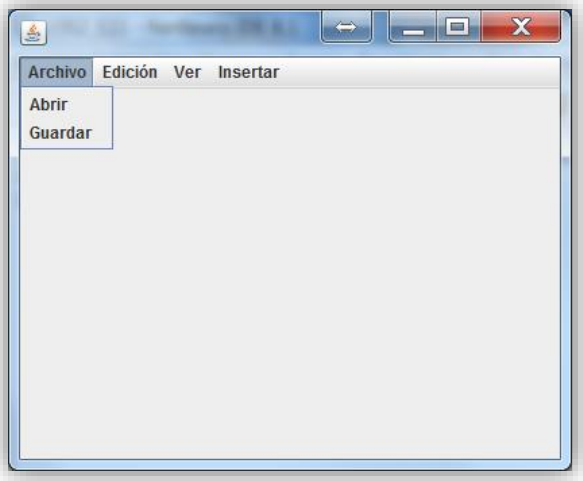


18. Usa ahora la propiedad *Text* de ambos JMenuItem para asignarles un texto. El primero tendrá el texto “Abrir” y el segundo el texto “Guardar”.



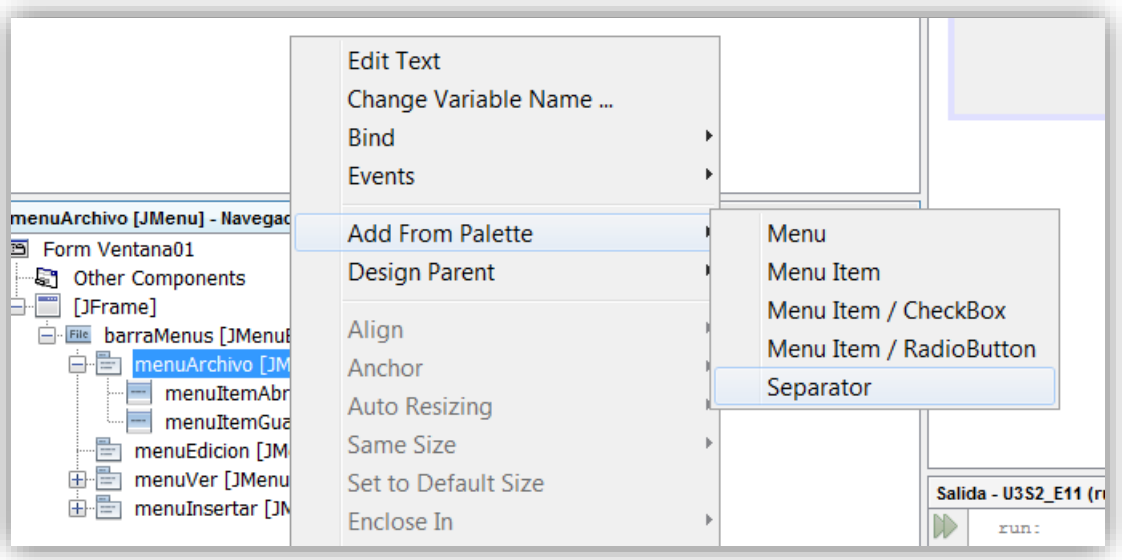
19. Ya podemos ejecutar el programa para ver qué es lo que se ha conseguido.

    Use el menú:



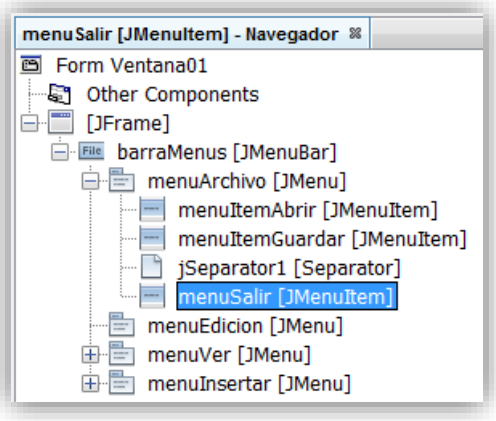
Observa como la opción Archivo se despliega mostrando dos submenús: Abrir y Guardar.

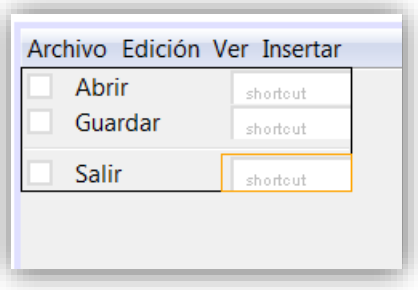
20. Seguiremos añadiendo elementos al menú. Ahora haga clic con el derecho sobre el elemento *menuArchivo* y añada un JSeparator.



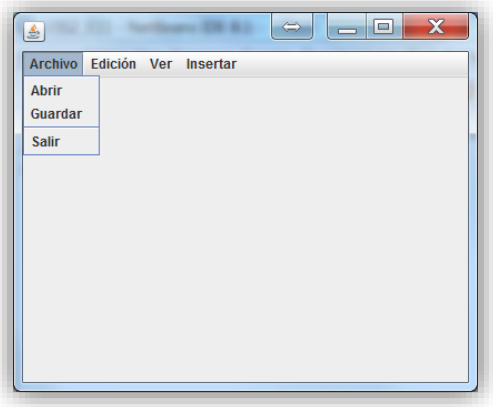
Los JSeparator son objetos que definen una separación entre las opciones de un menú. Cámbiele el nombre y llámelo “separador1”:

21. Añada un nuevo JMenuItem al menú Archivo y ponle el nombre menuSalir. El texto de esta opción será “Salir” (use su propiedad *text*) El aspecto del *Inspector* será el siguiente:





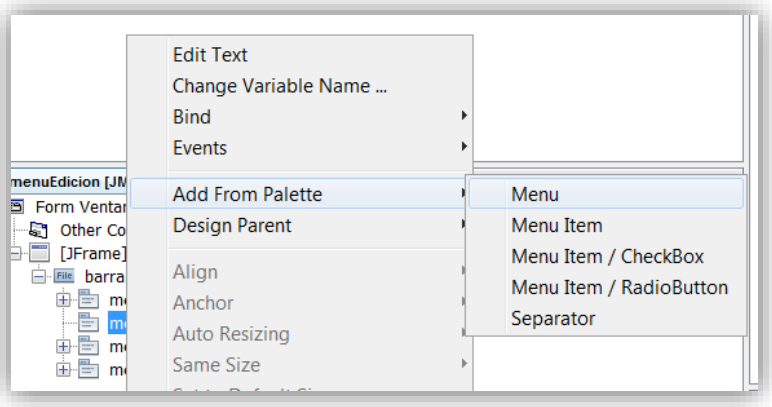
22. Ejecuta el programa y observa el contenido de la opción Archivo del menú:



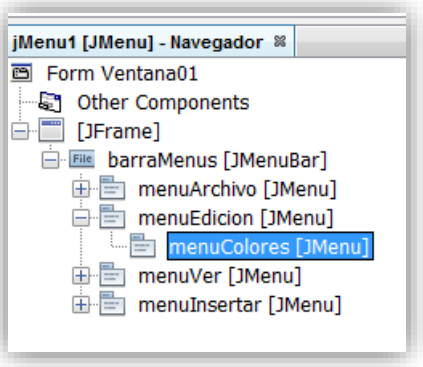
Observa el efecto que produce el separador.

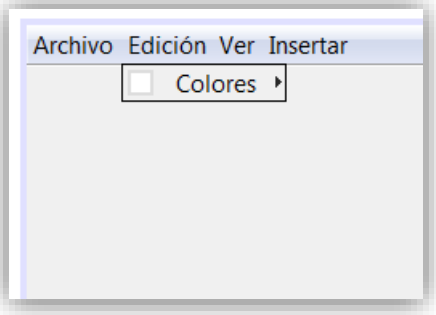
23. Un JMenu representa las opciones principales de la barra de menús. A su vez, un JMenu contiene JMenuItem, que son las opciones contenidas en cada opción principal, y que se ven cuando se despliega el menú.

    Sin embargo, un JMenu puede contener a otros JMenu, que a su vez contendrán varios JMenuItem. Usando el botón derecho del ratón y la opción "Añadir", añade un JMenu dentro de menuEdicion:

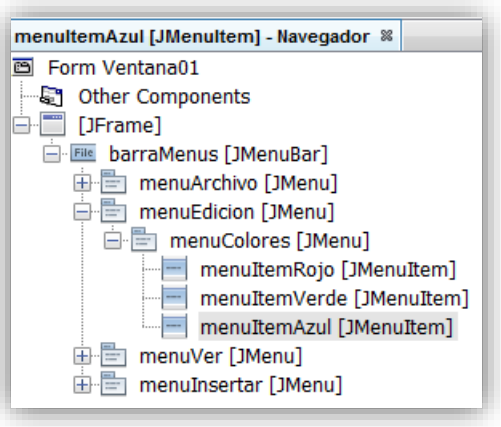


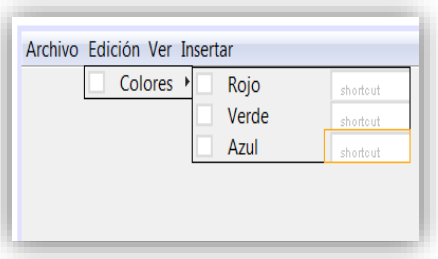
24. Llama al nuevo JMenu *menuColores* y asignale el texto "Colores".



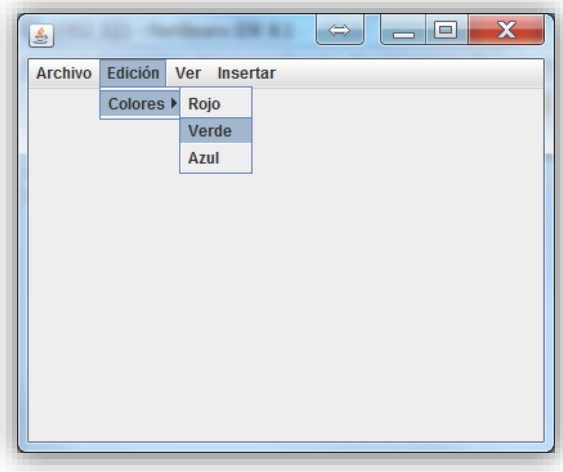


25. Ahora añade dentro del *menuColores* tres JMenuItem llamados respectivamente: *menuItemRojo*, *menuItemVerde*, *menuItemAzul*. Sus textos serán “Rojo”, “Verde” y “Azul”.





26. Ejecuta el programa y observa como ha quedado el menú Edición:



La opción Edición (JMenu) contiene una opción Colores (JMenu) que a su vez contiene las opciones Rojo, Verde y Azul (JMenuItems)

27. De nada sirve crear un menú si luego este no reacciona a las pulsaciones del ratón. Cada objeto del menú tiene un evento ActionPerformed que permite programar lo que debe suceder cuando se active dicha opción del menú.

28. Marque en el inspector el objeto menuItemRojo y acceda a su evento ActionPerformed. Dentro de él programe este sencillo código:

```
this.getContentPane().setBackground(Color.RED);
```

Este código cambia el color de fondo de la ventana a rojo.

29. Compruebe el funcionamiento de la opción “Rojo” del menú ejecutando el programa.

30. Programa tu mismo las opciones “Verde” y “Azul”.

```
this.getContentPane().setBackground(Color.GREEN);
```

```
this.getContentPane().setBackground(Color.BLUE);
```

CONCLUSIÓN

Las barras de menús son un conjunto de objetos de distinto tipo que se contienen unos a los otros:

La barra en sí está representada por un objeto del tipo JMenuBar.

La barra contiene opciones principales, representadas por objetos JMenu.

Las opciones principales contienen opciones que aparecen al desplegarse el menú. Esta opciones son objetos del tipo JMenuItem.

Un JMenu también puede contener otros JMenu, que a su vez contendrán JMenuItems.

También puede añadir separadores (JSeparator) que permiten visualizar mejor las opciones dentro de un menú.

Ejercicio Propuesto.
Teniendo en cuenta el enunciado del ejercicio 3-1.4 plantee un proyecto JAVA que permita ingresar temperaturas en grados Celsius y la convierta en Fahrenheit. Adjunta una carátula con tu Nombre y fecha. El ejercicio propuesto se entregará en la Sección / Tareas, de la plataforma, en formato .rar (archivo comprimido) que contenga los archivos del proyecto Java y la carátula con tus datos. (Caratula es libre, debe tener mínimamente Nombre y Apellido) "- Recuerda que el nombre del archivo a enviar debe comenzar con tu apellido.

Luego, utilizando winzip o winrar, comprimimos los archivos (Toda la carpeta del Proyecto) con el nombre de archivo que comience con su apellido y subimos el archivo comprimido a la plataforma junto a una caratula. La carátula con tus datos es libre, debe tener mínimamente Nombre y Apellido, puede ser un archivo de texto simple.
Podrías incluir el nombre de la UNL, Facultad, Carrera, Asignatura, etc.
Esto lo hacemos para que se vayan acostumbrando a identificar sus producciones de software, ya que siempre deberían tener la firma digital del autor.

Ejemplo de Carátula (Carátula.txt)

UNIVERSIDAD NACIONAL DEL LITORAL (UNL)
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS (FICH)

CARRERA: TECNICATURA APLICADA AL DISEÑO MULTIMEDIAL Y DE SITIOS WEB
ASIGNATURA: ELEMENTOS DE PROGRAMACIÓN
ALUMNO: PERES JUAN
TRABAJO: EJERCICIO PROPUESTO CORRESPONDIWNTE A LA UNIDAD Nº 2