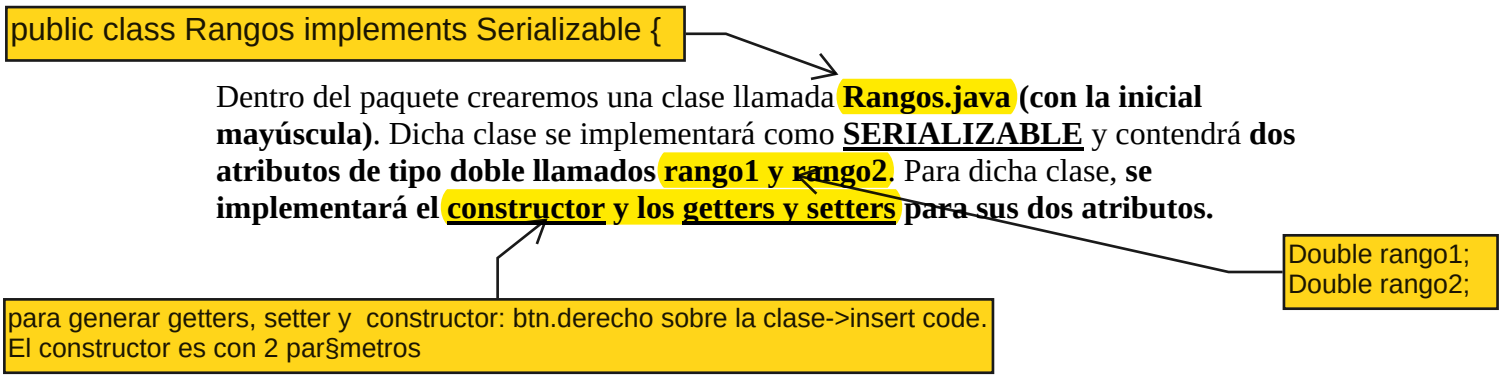
¿Qué te pedimos que hagas?

DESCRIPCIÓN DEL COMPONENTE A CREAR

Se pretende incorporar al componente JProgressBar la posibilidad de que permita definir tres intervalos sobre ella, de forma que el rótulo que muestre el porcentaje aparezca en color distinto dependiendo del intervalo en el que se sitúe el valor marcado. **TENER ESPECIAL CUIDADO CON LOS NOMBRES DE LOS ARCHIVOS A CREAR.**

FASE 1.- Definición de la clase para definir los rangos (Archivo Rangos.java)

Comenzaremos creando un **proyecto nuevo con el nombre ProgressBarExtended seguido de TUS INICIALES (por ejemplo ProgressBarExtendedAJC)**. Dentro del proyecto, crearemos **un paquete llamado progressbarextended (todo en minúscula).**

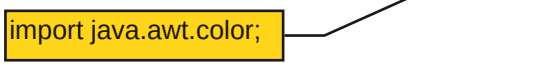


Dentro del paquete crearemos una clase llamada **Rangos.java (con la inicial mayúscula)**. Dicha clase se implementará como **SERIALIZABLE** y contendrá **dos atributos de tipo doble llamados rango1 y rango2**. Para dicha clase, **se implementará el constructor y los getters y setters para sus dos atributos.**

FASE 2.- Creación de la clase principal (Archivo ProgressBarExtended.java)

A continuación se creará el "JavaBeans Component" llamado ProgressBarExtended que **heredará de JProgressBar**, se implementará como **SERIALIZABLE** y presentará las siguientes propiedades:

- Un objeto del tipo Rangos denominado "rangos" (en minúscula).
- Tres variables de tipo Color para almacenar los colores correspondientes a cada rango denominadas **"color1", "color2"** y **"color3"**.



Deberán definirse los métodos getter y setter para las cuatro propiedades: **rangos, color1, color2 y color3**.

Dentro de la clase principal, declaramos esta clase:
public class RangoAlzanzado extends EventObject
Import ...util.EventObject;

A continuación se van a definir los elementos necesarios para trabajar con eventos. Para poder trabajar con eventos se hacen necesarias 4 cosas **TODAS ELLAS DENTRO DE LA CLASE PRINCIPAL**:

1. **Clase que implemente los eventos**.- Declararemos una clase nueva con su constructor denominada RangoAlcanzado que herede de EventObject.

import ....eventListener;

2. Deberemos **declarar una interfaz llamada RangoAlcanzadoListener** que heredará de EventListener y definirá 3 **métodos** que denominaremos **capturarZona1Alcanzada, capturarZona2Alcanzada y capturarZona3Alcanzada.** Dichos métodos recibirán como parámetro un evento del tipo RangoAlcanzado.

3. Deberán declararse **dos métodos: uno para agregar y otro para eliminar listeners** (addRangoAlcanzadoListener y removeRangoAlcanzadoListener). Ambos métodos recibirán como parámetro un objeto RangoAlcanzadoListener.

4. **MUY IMPORTANTE**: Una vez hecho lo anterior, tenemos que **añadir** un atributo nuevo que será **un objeto de tipo RangoAlcanzadoListener** al que llamaremos **"listener"**.

ááATENTO!!! en estos m®todos es donde hay que meter el a¶adir y borrar el c· digo para listener cuando se alcance el rango.

addRango-> this.listener=ral; (nombre del parametro que paso);

removeRango-> this.listener=null;

A continuación, colocaremos **el siguiente código en el constructor**:

áCUIDADO!! Hay que anular (//) las l²neas de c· digo de los m®todos: addPropertyChangeListener y removerPropertyChangeListener

```
public ProgressBarExtended() {

      // Definimos los intervalos

        rangos = new Rangos(50, 75);


      // Añadimos ChangeListener y redefinimos el método stateChanged

        this.addChangeListener(new ChangeListener() {

            @Override

            public void stateChanged(ChangeEvent e) {

                comprobarZona();

            }

        });
}
```

import swing.event.ChangeEvent

Seguidamente, definiremos el método **comprobarZona**. En dicho método, haremos las siguientes comprobaciones **anidándolas correctamente**:

1. Si el valor del componente (this.getValue()) es **menor que el valor rango1**, (recordar que dicha propiedad está dentro del objeto rangos), colocaremos como **color del texto (foreground)** el **color1**.

    - Si el valor del componente es **igual al valor de la propiedad rango1 MENOS UNA UNIDAD**, se comprobará **si el listener es distinto de null**, en caso de ser distinto de null, llamaremos al método del listener **capturarZona1Alcanzada** con la siguiente sintaxis: (tener en cuenta que se entra en Zona1 cuando se está por debajo del rango1)

```
listener.capturarZona1Alcanzada(new RangoAlcanzado(this));
```

2. Si el valor del componente es **mayor o igual que rango1 y menor o igual que rango2**, colocaremos como **color del texto (foreground)** el **color2**.

    - Si el valor del componente es **igual al valor de la propiedad rango1 o igual al valor de la propiedad rango2**, se comprobará **si el listener es distinto de null,** en caso de ser distinto de null, se llamará al método del listener **capturarZona2Alcanzada** en la misma forma que antes. (Tener en cuenta que se entra en Zona2 en el momento en que el valor del componente se iguala al valor rango1 (por abajo) y al rango2 (por arriba)).

3. En caso de que no se cumpla lo indicado en 1 ni lo indicado en 2, significa que el valor del componente es superior a rango2 y, por tanto, colocaremos como **color del texto (foreground)** el **color3**.

    - Si el valor del componente es **igual al valor de la propiedad rango2 MÁS UNA UNIDAD**, se comprobará **si el listener es distinto de null,** en caso de ser distinto de null, llamaremos al método del listener **capturarZona3Alcanzada** igual que en los casos anteriores. (Tener en cuenta que se entra en Zona3 en el momento en que el valor del componente supera el valor de rango2).

FASE 3.- Creación de un editor de propiedades personalizado (Archivo ProgressBarPanel.java)

A continuación vamos a crear un editor personalizado para la propiedad **"rangos"**. Para ello, añadimos al paquete de nuestro proyecto un JPane**l** Form y le ponemos como nombre **ProgressBarPanel**. El aspecto del mismo es el siguiente:

Editor personalizado de rangos de la JProgressBar.

Los JSlider se llamarán **JSliderNivel1** y **JSliderNivel2 y deberán quedar inicializados DURANTE EL DISEÑO a los valores 50 y 75 respectivamente**. Las **etiquetas de la derecha** se llamarán **JLabelNivel1** y **JLabelNivel2 y mostrarán por defecto los textos 50% y 75%**.

Debemos programar un método público llamado **getSelectedValue** que devuelva el valor de la propiedad de tipo Rangos seleccionada en el panel. El código de dicho método es el siguiente:

```
  public Rangos getSelectedValue(){

        return new Rangos(jSliderNivel1.getValue(),
jSliderNivel2.getValue());

  }
```

También **se deberá programar un método para mostrar en las etiquetas (se puede llamar <u>mostrarValores</u>)**, para mostrar en las JLabelNivel1 y JLabelNivel2 los valores de los JSliderNivel1 y JSliderNivel2 respectivamente en formato de porcentaje (valor numérico seguido del carácter %).

Para evitar que el valor del rango2 sea inferior al valor del rango1 y que el valor del rango1 sea superior al rango2, **deberemos programar los métodos stateChanged del JSliderNivel1 y JSliderNivel2.**

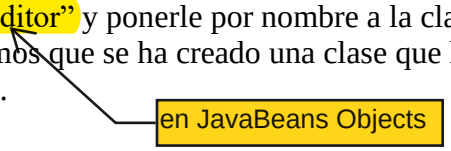
<u>En el método stateChanged del JSliderNivel1</u>, comprobaremos si el valor del JSliderNivel1 es mayor o igual que el valor del JSliderNivel2. En dicho caso, asignaremos al JSliderNivel2 el valor del JSliderNivel1. Para finalizar dicho método, **llamaremos al método mostrarValores**.

<u>En el método stateChanged del JSliderNivel2</u>, comprobaremos si el valor del JSliderNivel2 es menor o igual que el valor del JSliderNivel1. En dicho caso, asignaremos al JSliderNivel1 el valor del JSliderNivel2. Para finalizar dicho método, **llamaremos al método mostrarValores**.

<u>Fase 4.- Creación de la clase ProgressBarPropertyEditor para relacionar el editor con la propiedad</u>

<u>(Archivo ProgressBarPropertyEditor.java)</u>

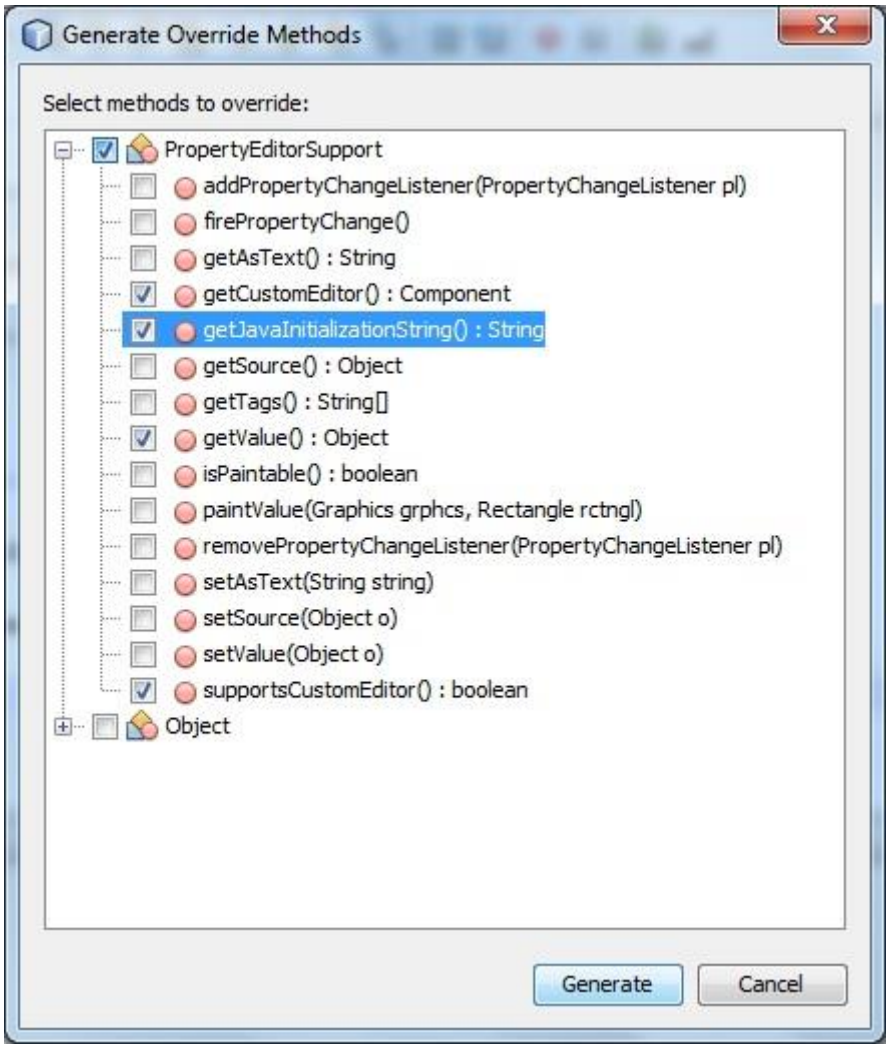
A continuación tenemos que crear una nueva clase que herede de PropertyEditorSupport. Para ello, acceder al proyecto y agregar al paquete un "PropertyEditor" y ponerle por nombre a la clase **ProgressBarPropertyEditor**. Observaremos que se ha creado una clase que hereda de PropertyEditorSupport y su constructor.



Delante del constructor vamos a colocar una única propiedad para dicha clase denominada "**rangoPanel**" que **será un objeto de tipo ProgressBarPanel**.

En el constructor de la clase ProgressBarPropertyEditor **inicializamos el objeto "rangoPanel" llamando con new al constructor de la clase ProgressBarPanel**.

Ahora vamos a redefinir cuatro de los métodos de la clase "PropertyEditorSupport", para ello, pulsamos botón derecho "Insert Code…" y a continuación "Override Method…". Marcamos los métodos que se muestran en la imagen (getCustomEditor, getJavaInitializationString, getValue y supportsCustomEditor):



Métodos de la clase PropertyEditorSupport a redefinir.

A continuación se indica qué hace cada uno de los 4 métodos y qué código habrá que colocar en cada uno de ellos:

1.- **Método supportsCustomEditor**.- El código de este método es simplemente devolver true, ya que indica si se está utilizando o no un editor personalizado para la propiedad. Como sí se está utilizando, devolvería true.

2.- **Método getCustomEditor**.- El código de este método sería devolver el objeto de nombre **rangoPanel** que hemos declarado como propiedad de la clase.

3.- **Método getValue**.- Este método devolvería el resultado de la llamada al método getSelectedValue del objeto **rangoPanel** que hemos declarado como propiedad de la clase. Recordemos que ese método getSelectedValue lo implementamos anteriormente para devolver la propiedad de tipo Rangos seleccionada en el panel.

4.- **Método getJavaInitializationString**.- El código del método **getJavaInitializationString** deberá ser el siguiente **(tener especial cuidado al codificarlo)**:

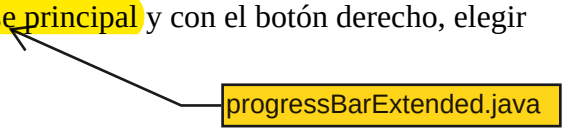
```
@Override

public String getJavaInitializationString(){

Rangos rangos=(Rangos)getValue();

return ("new progressbarextended.Rangos(" + rangos.getRango1() + ", "
+ rangos.getRango2() + ")");

}
```

Lo que hace el método **getJavaInitializationString** es devolver la cadena de código necesaria para que se cree la propiedad rangos con los valores seleccionados por el usuario en el editor personalizado (progressbarextended es el nombre del paquete).

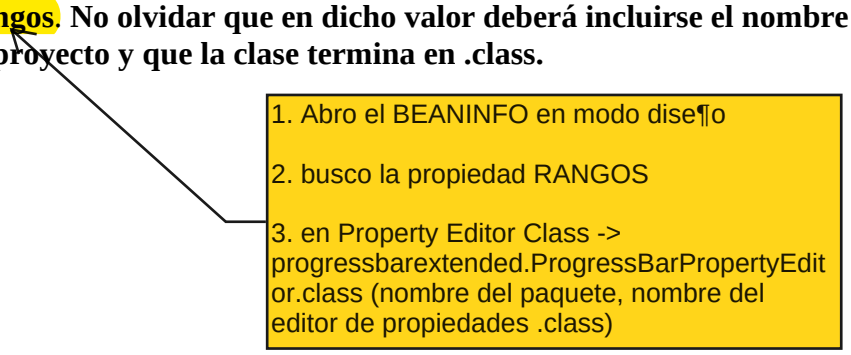
Fase 5.- Creación del BeanInfo para conectar la clase que hereda de PropertyEditorSupport a la propiedad

(Archivo ProgressBarExtendedBeanInfo.java)

A continuación creamos el BeanInfo que relacionará la clase basada en PropertyEditorSupport con la propiedad para permitirnos usar el editor personalizado. Para crearlo, deberemos seleccionar la clase principal y con el botón derecho, elegir "BeanInfo Editor".



Una vez creado, asignaremos el valor de la clase del editor de propiedades para la propiedad **rangos**. **No olvidar que en dicho valor deberá incluirse el nombre del paquete del proyecto y que la clase termina en .class.**



Fase 6.- Obtener el componente, agregarlo a la paleta y crear un proyecto para realizar una prueba de su funcionamiento

Llegados a este punto, se puede **ejecutar "Clean and Build"** o "Limpiar y Construir" sobre la clase ProgressBarExtended.java y agregar el componente a la categoría de la paleta de componentes que consideremos oportuna.

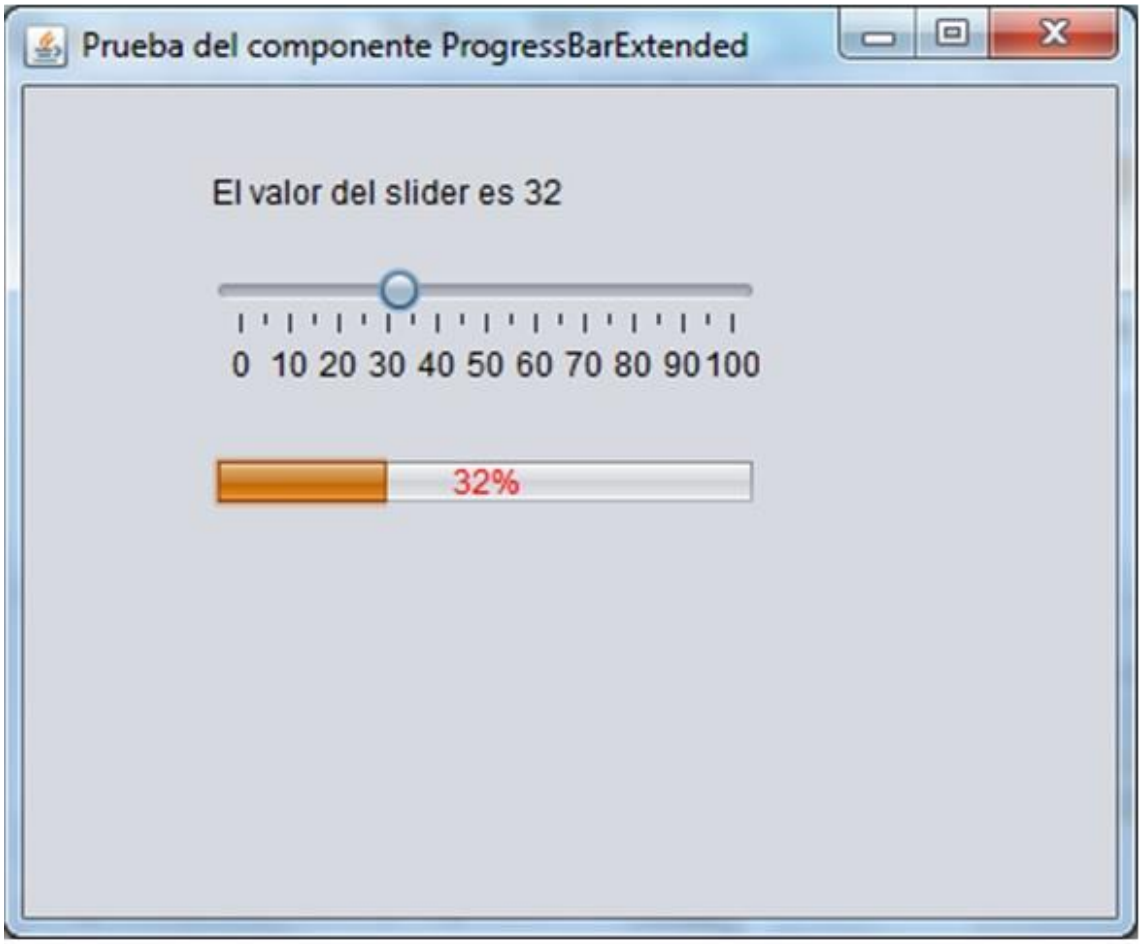
ProgresBarExtended-> add to palette

Si quiero crear una categor²a propia-> Tool -> palette->swing-> New category

Una vez agregado a la paleta, crearemos una "Java Application" para probar el componente. En dicha aplicación crearemos un paquete y dentro de éste colocaremos el formulario principal.

Crear un nuevo proyecto y a¶adir un JFrame

En dicho formulario principal incluiremos una etiqueta, un JSlider y un componente de tipo ProgressBarExtended. El aspecto del formulario será el siguiente:



Aplicación de prueba realizada para comprobar el funcionamiento del componente ProgressBarExtended.

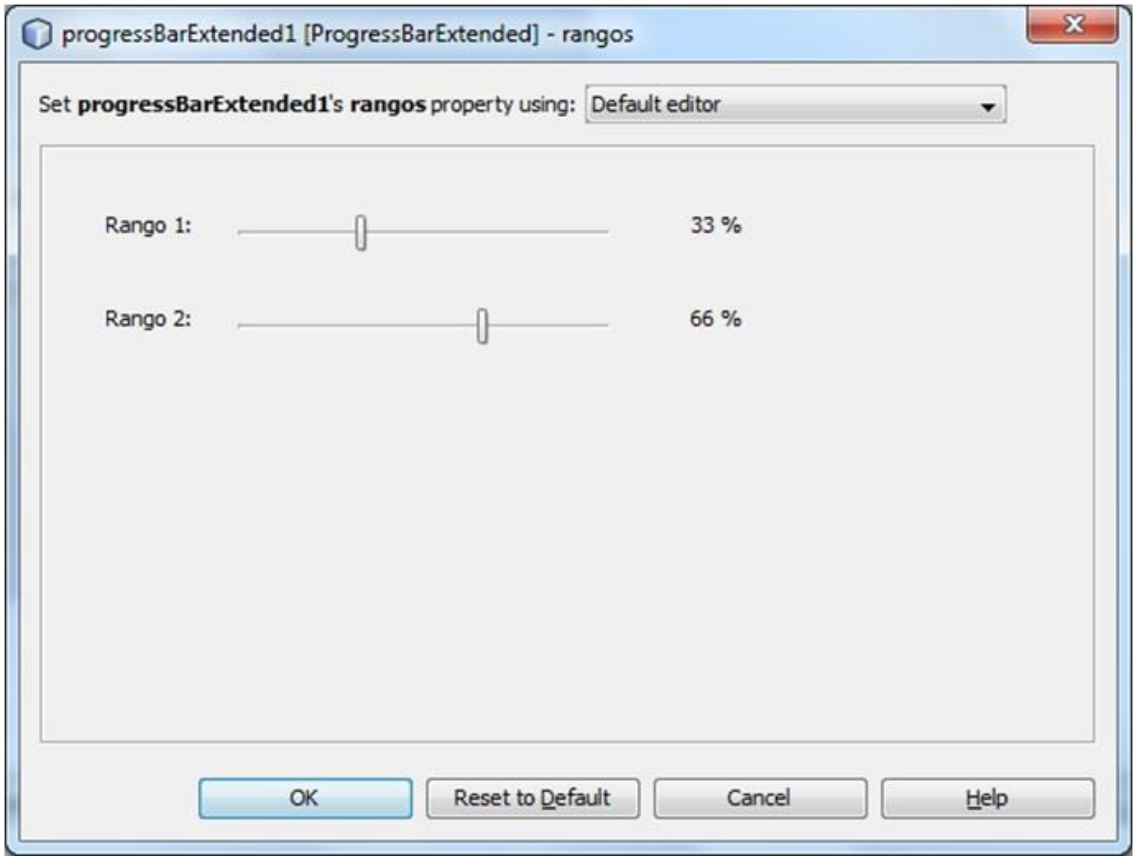
Para que la aplicación tenga el aspecto que se muestra se deberá hacer lo siguiente:

En el JSlider se deberán configurar las siguientes propiedades:

1. Marcaremos las propiedades **“paintLabels”, “paintTicks” y “paintTrack”**.
2. La propiedad **MajorTickSpacing** se establecerá a **10** y la propiedad **MinorTickSpacing** a **5**.
3. La propiedad **Maximum** se establecerá a **100**, la propiedad **Minimum** a **0** y la propiedad **Value** a **50**.

En el componente ProgressBarExtended configuraremos las siguientes propiedades:

1. Marcaremos la propiedad **stringPainted**.
2. Estableceremos los colores que consideremos oportunos para las propiedades **color1, color2 y color3.**
3. Por último, **accederemos a la propiedad rangos y comprobaremos el funcionamiento del editor personalizado** para la misma. Estableceremos los valores para el **rango1** y el **rango2** que consideremos oportuno.



Editor personalizado de la propiedad Rangos.

Deberemos programar un método al que llamaremos establecerValor que colocará en la etiqueta la cadena para mostrar el valor del JSlider y que establecerá el valor de la ProgressBarExtended al valor señalado por el JSlider.

en el source del formulario principal

A continuación colocaremos **dos llamadas al método establecerValor**:

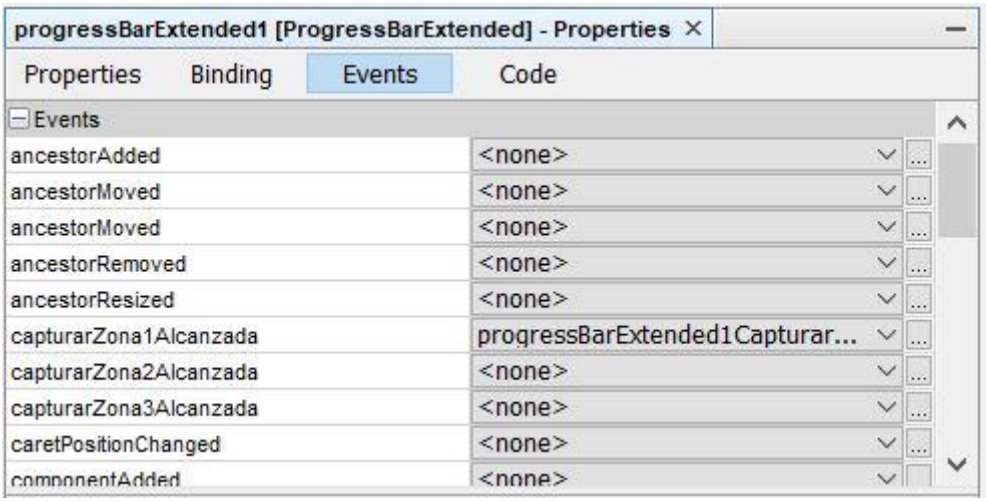
1. La primera **tras la llamada al método initComponents**.
2. Seguidamente, **programaremos el método stateChanged del JSlider** y como única línea de dicho método colocaremos la llamada al método establecerValor.

1. Design
2. Slider ->Eventos
3. Change

Para finalizar, podemos programar los métodos **capturarZona1Alcanzada, capturarZona2Alcanzada y capturarZona3Alcanzada** para que muestren un mensaje empleando un JOptionPane cada vez que se alcance cada zona. El siguiente mensaje se mostrará cuando se alcance la Zona 1:

Mensaje que indica que se ha alcanzado la Zona 1.

En principio, en la lista de eventos correspondientes al componente progressBarExtended que hemos colocado deberían aparecer los eventos capturarZona1Alcanzada, capturarZona2Alcanzada y capturarZona3Alcanzada disponibles para ser programados tal como se muestra en la siguiente figura.



Eventos del componente ProgressBarExtended en el programa de prueba.

Si no aparecen, habría que añadir el código siguiente en el constructor de la clase del programa de prueba (en este código suponemos que el componente que se ha colocado se llama progressBarExtended1, en caso de que se haya cambiado el nombre, habrá que adaptarlo):

```
progressBarExtended1.addRangoAlcanzadoListener(new
progressbarextended.ProgressBarExtended.RangoAlcanzadoListener() {

@Override

public void
capturarZona1Alcanzada(progressbarextended.ProgressBarExtended.RangoAl
canzado evt) {

// Código necesario para mostrar el mensaje de aviso al alcanzar la
Zona 1
 }

@Override

public void
capturarZona2Alcanzada(progressbarextended.ProgressBarExtended.RangoAl
canzado evt) {
// Código necesario para mostrar el mensaje de aviso al alcanzar la
Zona 2

}

@Override

public void
capturarZona3Alcanzada(progressbarextended.ProgressBarExtended.RangoAl
canzado evt) {

// Código necesario para mostrar el mensaje de aviso al alcanzar la
Zona 3
}
});
```

Recursos necesarios y recomendaciones

Se recomienda realizar las prácticas propuestas en la Unidad para que se tengan bien claros los pasos a seguir en el proceso de desarrollo de componentes visuales.

Como recurso se empleará el IDE de NetBeans tanto para el desarrollo del componente como del programa de prueba de funcionamiento del mismo.

Indicaciones de entrega

Una vez realizada la tarea tendrás que comprimir los archivos que has generado y subirlos a la plataforma. La estructura de archivos a entregar dentro del archivo comprimido es como sigue:

- Dentro de un directorio de nombre **programa**:
  - El directorio con el proyecto NetBeans en el que hayas creado el componente.
  - El directorio con el proyecto NetBeans de prueba del componente.

El envío se realizará a través de la plataforma de la forma establecida para ello, y el archivo se nombrará siguiendo las siguientes pautas:

Apellido1_Apellido2_Nombre_DI03_Tarea

Asegúrate que el nombre no contenga la letra ñ, tildes ni caracteres especiales extraños. Así por ejemplo la alumna **Begoña Sánchez Mañas para la tercera unidad del MP de DI**, debería nombrar esta tarea como...

Sanchez_Manas_Begona_DI03_Tarea