INTRODUCCIÓN

En este documento voy a explicar la creación de un componente visual, que pueda ser integrado en la paleta de componentes de Netbeans. El objetivo principal de este tema, es aprender a programar **software reutilizable**. Es decir, el componente, una vez integrado en la paleta de Netbeans, podrá ser utilizado directamente en nuevos proyectos. Este tipo de desarrollo tiene varias ventajas:

1. Solo tenemos que programar el componente una vez y lo podemos utilizar directamente en varios proyectos (ahorro de tiempo además de evitar el "copy&paste").
2. Si hay un error en el componente, es suficiente corregirlo en el proyecto del componente y de esta manera podríamos actualizar el resto de proyectos que utilizan el componente sin necesidad de tocarlo.
3. Durante la fase de pruebas, sólo necesitamos probar el componente una vez.

ELECCIÓN DEL COMPONENTE BASE

Un componente visual como los que vamos a desarrollar en este tema, puede ser realizado desde cero, sin tener como base ningún otro componente. De hacerlo así, tendríamos que realizar un desarrollo muy laborioso ya que sería necesario encargarse de programar todo el componente, su manera de mostrarse en pantalla, sus eventos, etc. Por esta razón, nosotros partiremos siempre de un componente de Swing ya existente, y utilizaremos la herencia para extenderlo de la manera que nos interese.

Debido a lo anterior, en la primera fase debemos elegir el componente del que vamos a extender. En primera instancia, si nuestro componente va a mostrar un texto, extenderíamos de *JLabel*, si es un componente que va de alguna manera a permitir la introducción de texto, podríamos elegir un *JTextField*. Resumiendo, tenemos que elegir el componente base que mejor nos encaje para aprovechar toda su funcionalidad base y empezar a construir a partir de ahí.

CONCEPTO DE JAVABEAN

Una vez elegido el componente del que vamos a heredar, podemos comenzar ya con el desarrollo propiamente dicho. Lo primero que tenemos que tener en cuenta es que un componente, para poder ser integrado en la paleta de Netbeans, debe de ser un **JavaBean**. Un JavaBean no es más que una clase que cumple tres condiciones:

1. Implementa Serializable.
2. Tiene un constructor sin parámetros.

3. Los getters y setters de sus propiedades están nombrados de la manera estándar (como los genera Netbeans).

Teniendo en cuenta lo anterior, los primeros pasos serán hacer que nuestra clase implemente Serializable y crear un constructor sin parámetros.

A continuación debemos de identificar que propiedades va a tener nuestro componente. Las propiedades serán valores configurables del componente a través del editor de propiedades de NetBeans. Por ejemplo, el componente JButton de Swing, tiene una propiedad text, utilizada para establecer el texto que aparece en el botón (ver Ilustración 1). Esta propiedad text, puede ser editada desde el editor de propiedades de Netbeans cuando arrastramos el componente a nuestro interfaz gráfico.

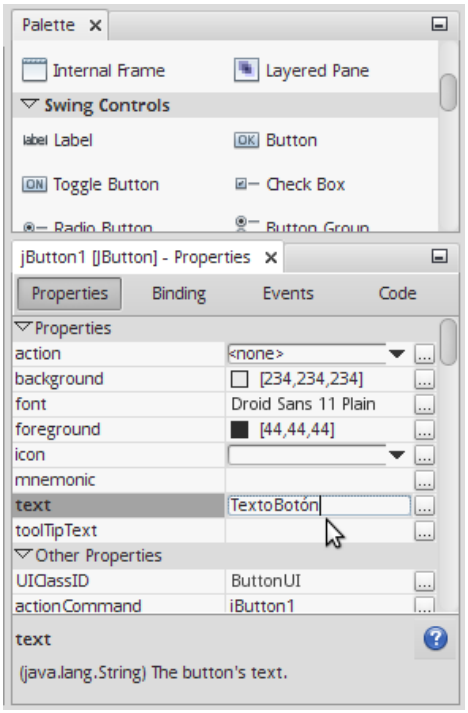

Ilustración 1: Edición de la propiedad text de un JButton

De la misma manera funcionan nuestras propiedades. Tan solo incluyendo el atributo en la clase de nuestro componente y creando su getter y su setter, el editor de Netbeans es capaz de mostrarnos la propiedad y de permitir su edición. Veamos un ejemplo.

Ejemplo práctico 1: Creación de un componente que permita introducir un texto. El componente tendrá una propiedad de tipo Color, que será el color de fondo del componente cuando el usuario haya introducido un número de caracteres superior al número indicado por una segunda propiedad de tipo entero.

En el ejemplo indicado anteriormente, vemos claramente que se trata de un componente que permite la entrada de texto. Así pues, extenderemos el componente de JTextField. Por otra parte, el componente tiene dos propiedades, una de tipo Color y otra de tipo entero. En este momento ya estamos en disposición de escribir el esqueleto de nuestro componente, cumpliendo las condiciones que tiene que tener para ser un JavaBean.

```
public class ComponenteEjemplo extends JTextField implements Serializable
{
   //Color de fondo cuando se llegue al nÃºmero de caracteres indicado en numCaracteres
   private Color colorFondo;
   private int numCaracteres;

   public ComponenteEjemplo()
   {
   }

   public Color getColorFondo() {
      return colorFondo;
   }

   public void setColorFondo(Color colorFondo) {
      this.colorFondo = colorFondo;
   }

   public int getNumCaracteres() {
      return numCaracteres;
   }

   public void setNumCaracteres(int numCaracteres) {
      this.numCaracteres = numCaracteres;
   }
}
```

Como se puede ver en el trozo de código anterior, nuestro componente ya cumple las condiciones para ser un JavaBean. Sólo con ese trozo de código, ya podríamos añadir el componente a la paleta de componentes de Netbeans, e insertar nuestro componente en cualquier proyecto (aunque todavía no tendría ninguna funcionalidad). Para ello, debemos seguir los siguientes pasos:

1. Compilar el proyecto del componente (botón derecho sobre el proyecto, y pulsar Clean and Build).
2. Añadir el componente a la paleta. Para hacerlo, seleccionamos la clase con el componente y con el botón derecho pulsamos en Tools>Add to palette... . Nos aparecerá una ventana para elegir la categoría en la que queremos meter el componente y pulsamos Aceptar.

Una vez realizados estos pasos, podemos crear un proyecto de prueba para introducir el componente y ver que podemos editar las propiedades sin hacer nada más. Es importante tener en cuenta que Netbeans es capaz de mostrar y permitir la edición de propiedades automáticamente sólo para los tipos básicos (int, double, String, Color, boolean). En el caso de que nuestra propiedad sea de un tipo más complejo, tendremos que realizar un editor de propiedades personalizado (lo veremos más adelante).

FUNCIONALIDAD DEL COMPONENTE

Una vez en este punto, con el componente creado, y desde la paleta del Netbeans ya se pueden editar sus propiedades, solo nos queda programar la funcionalidad. En este caso concreto, debemos ser capaces de saber en cada momento cuantos caracteres hay introducidos y cambiar el color de fondo en consecuencia. Para ello escribimos el siguiente código en el constructor:

```
public ComponenteEjemplo()
{
    super();
    //Salvamos el color que tenÃa el componente por defecto
    colorPorDefecto = getBackground();
    super.getDocument().addDocumentListener(new DocumentListener()
    {
      @Override
      public void insertUpdate(DocumentEvent e){
        analizaContenido();
      }

      @Override
      public void removeUpdate(DocumentEvent e){
        analizaContenido();
      }

      @Override
      public void changedUpdate(DocumentEvent e){
        analizaContenido();
      }

      private void analizaContenido(){
        if (getText().length()>=numCaracteres)
          setBackground(colorFondo);
        else
          setBackground(colorPorDefecto);
      }
    });
  }
```

Como podemos observar en el código anterior, utilizamos las dos propiedades para modificar el color de fondo del componente según el número de caracteres introducidos en el mismo. Hemos creado también una variable interna para almacenar el color de fondo por defecto y poder restablecerlo en el caso de que el usuario borre caracteres.

CREACIÓN DE UN EDITOR DE PROPIEDADES PERSONALIZADO

En el caso anterior, las dos propiedades del componente eran propiedades simples, que el Netbeans nos permitía editar desde la paleta sin necesidad de hacer nada más. Sin embargo, puede darse el caso de que tengamos que crear propiedades más complejas, en las que sería necesario un editor de propiedades personalizado para editarlas. Imaginemos el siguiente caso:

Ejemplo práctico 2: Creación de un componente que permita introducir un texto. El componente tendrá una propiedad denominada colorComponente, que permitirá establecer el color de fondo del componente y el color del texto del componente cuando el usuario haya introducido un número de caracteres superior al número indicado por una segunda propiedad de tipo entero.

Como se puede observar en el enunciado anterior, la propiedad colorComponente, es una propiedad compleja, formada por dos colores. En una situación como ésta, no queda otra opción que realizar un editor personalizado, de manera que cuando el programador que use este componente vaya a editar la propiedad colorComponente, aparezca un interfaz adecuado para seleccionar dos colores. Veamos paso por paso como se haría.

Paso 1 – Crear la clase necesaria para almacenar la propiedad personalizada

En este problema tenemos que almacenar dos colores. La clase para la propiedad podría tener esta forma:

```
public class ClaseDosColores  implements Serializable
{
   private Color colorFondo;
   private Color colorTexto;

   public ClaseDosColores(Color colorFondo, Color colorTexto){
      this.colorFondo = colorFondo;
      this.colorTexto = colorTexto;
   }

   public Color getColorFondo(){
      return colorFondo;
   }

   public Color getColorTexto(){
      return colorTexto;
   }
}
```

Paso 2 – Definir la propiedad en el componente, con el tipo que hemos creado anteriormente

La propiedad en el componente será ahora de tipo ClaseDosColores. La definimos con su getter y setter:

```
public class ComponenteEjemplo2 extends JTextField implements Serializable
{
   private ClaseDosColores colores;
   private int numCaracteres;

   public ComponenteEjemplo2() {
```

```
    }

    public ClaseDosColores getColores(){
        return colores;
    }

    public void setColores(ClaseDosColores colores){
        this.colores = colores;
    }

    public int getNumCaracteres() {
        return numCaracteres;
    }

    public void setNumCaracteres(int numCaracteres) {
        this.numCaracteres = numCaracteres;
    }
}
```

Podemos ver que al igual que en el primer ejemplo, nuestro componente sigue cumpliendo las condiciones para ser un JavaBean. Es importante observar que la clase en la que definimos la propiedad también implementa Serializable. Es necesario que lo haga ya que sino nuestro componente, al incluir una propiedad de este tipo, dejaría de ser Serializable y por tanto no cumpliría las condiciones para ser un JavaBean.

Si agregásemos el componente según está a la paleta, podríamos editar la propiedad numCaracteres, pero sería imposible editar la propiedad colores.

Paso 3 – Crear el panel que va a aparecer en el editor de propiedades personalizado

En este paso, vamos a diseñar el panel que va a servir para introducir el valor de la propiedad en tiempo de diseño (Ilustración 2). Para no complicarnos con la introducción de los colores, recogeremos estos por su código en hexadecimal. Es importante indicar que el panel no tiene que tener botones de aceptar/cancelar ni ninguna otra cosa extra que no sea lo estrictamente necesario para recoger el valor de la propiedad.

Ilustración 2: Panel para recoger la propiedad colores

Dentro del panel, debemos crear un método que utilizaremos posteriormente para devolver el valor de la propiedad introducido por el usuario programador.

```
public ClaseDosColores getPropiedadSeleccionada()
   {
      if (!textFieldColorFondo.getText().equals("") && !textFieldColorText.getText().equals(""))
      {
         Color colorFondo = Color.decode(textFieldColorFondo.getText());
         Color colorTexto = Color.decode(textFieldColorText.getText());
         return  new ClaseDosColores(colorFondo,colorTexto);
      }
      else
         return null;
   }
```

Es importante destacar que este método tiene que devolver una clase del mismo tipo que el hayamos usado para definir la propiedad que estamos editando.

Paso 4 – Crear la clase PropertyEditorSupport

Después de diseñar un panel capaz de recoger la propiedad, tenemos que crear una clase que extienda de PropertyEditorSupport. Esta es la clase que va a indicar al NetBeans que tenemos un editor de propiedades personalizado y como usarlo. Vamos a ver una implementación de esta clase para este ejemplo concreto y a explicar cada uno de sus métodos.

```
public class ColoresPropertyEditorSupport extends PropertyEditorSupport
{
   private ColoresPanel coloresPanel = new ColoresPanel();
   @Override
   public boolean supportsCustomEditor()
   {
      return true;
   }

   @Override
   public Component getCustomEditor()
   {
      return coloresPanel;
   }

   @Override
   public Object getValue()
   {
      return coloresPanel.getPropiedadSeleccionada(); //To change body of generated methods, choose Tools
| Templates.
   }
```

```
    @Override
    public String getJavaInitializationString()
    {
        Color colorFondo = coloresPanel.getPropiedadSeleccionada().getColorFondo();
        Color colorTexto = coloresPanel.getPropiedadSeleccionada().getColorTexto();
        return "new componenteejemplo.ClaseDosColores(new java.awt.Color("+
                colorFondo.getRGB()+"),new java.awt.Color("+colorTexto.getRGB()+"))";
    }
}
```

Explicación de los métodos:

- supportsCustomEditor. Este método será llamado por Netbeans para preguntar si existe o no un editor de propiedades personalizado. Debemos devolver true en el caso de que lo tengamos.
- getCustomEditor. Este método devolverá el panel que hemos creado para editar la propiedad. Es decir, Netbeans preguntará al método supportsCustomEditor si hay un editor personalizado, y en caso de que lo haya, llamará a este método para obtener el panel que tendrá que mostrarle al usuario. Como se puede ver en la implementación, devolvemos una instancia del panel.
- getValue(). Una vez que NetBeans muestra el panel para permitir al usuario programador la edición de la propiedad y se pulsa el botón Aceptar, se llamará este método para obtener el valor de la propiedad del panel. Es muy importante que este método devuelva un objeto del tipo de la propiedad (ClaseDosColores en este caso).
- getJavaInitializationString(). Este es el método más complicado de todos. Sirve para ayudar al Netbeans en la generación de código necesaria para inicializar la propiedad en tiempo de ejecución. Cuando se arrastra un componente a un formulario, Netbeans genera un trozo de código dentro del método initComponents. Si únicamente arrastramos el componente y lo incluimos en un formulario, generararía una llamada al constructor del componente.

  ```
  componenteEjemplo22 = new componenteejemplo.ComponenteEjemplo2();
  ```

  Si además de arrastrar el componente, cambiamos el valor de una propiedad en la vista de diseño, nos generaría una llamada al setter correspondiente para que cuando el programa sea ejecutado, esa propiedad tenga el valor deseado por el programador. Imaginemos que cambiamos la propiedad text:

  ```
  componenteEjemplo22.setText("Hola");
  ```

En el caso de que la propiedad editada sea la que acabamos de hacer, tendría que funcionar de la misma manera, y generarnos una llamada al setter para inicializar el valor de la propiedad según lo que hayamos seleccionado en el panel. El problema está en que el setter de nuestra propiedad recibe una clase creada por nosotros (ClaseDosColores) y por supuesto el Netbeans no tiene ni idea de como inicializarla a partir de la información del panel. Tenemos por tanto que ayudarlo. El método getJavaInicializationString devuelve un String (la parte subrayada en amarillo abajo) con el código que habría que meter entre los paréntesis de la llamada al setter. Concretamente en nuestro caso, Netbeans generaría algo de este estilo:

```
componenteEjemplo22.setColores(new componenteejemplo.ClaseDosColores(new java.awt.Color(-65281),new java.awt.Color(-16711936)));
```

Como se puede observar en el código anterior, he utilizado el código RGB del color para inicializar los dos colores seleccionados en el panel. Desgraciadamente, no existe un mecanismo sencillo para orientaros sobre como programar este método. Simplemente hay que conseguir crear un String que encaje perfectamente en el setter de nuestra propiedad y que sea capaz de inicializar el valor de la propiedad según lo que se haya seleccionado en el editor de propiedades.

De esta forma, una vez arrancado el proyecto de prueba, el valor de la propiedad tomaría los valores deseados y dentro del componente podríamos trabajar con esos valores para programar la funcionalidad deseada.

Paso 5 – Crear el BeanInfo del componente

Este es el último paso en la creación del editor de propiedades personalizado. Sirve indicarle al NetBeans a que propiedad corresponde el PropertyEditorSupport que hemos creado.

Para ello, pulsamos encima del componente con el botón derecho y seleccionamos BeanInfo Editor... . De esta forma aparecerá una nueva clase en nuestro proyecto. La abrimos y nos situamos en vista de diseño. Buscamos la propiedad que hemos creado (colores en este caso), e indicamos la clase PropertyEditorSupport que hemos hecho anteriormente (ver Ilustración 3).

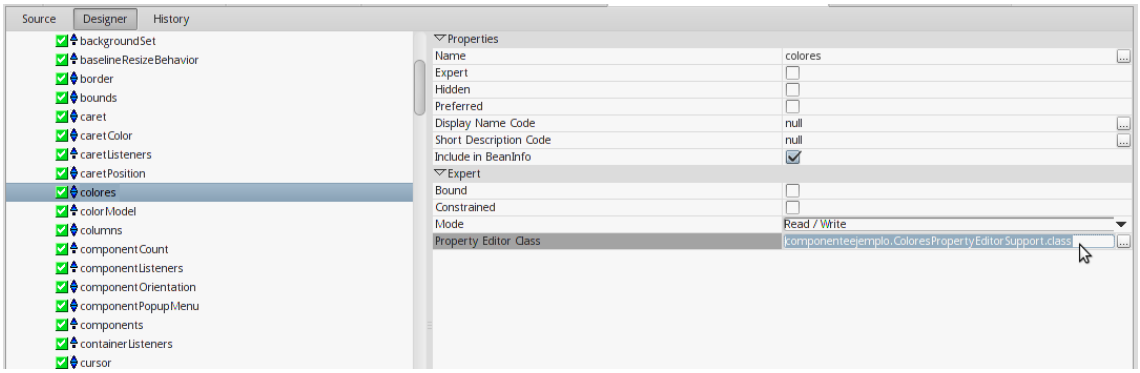

Ilustración 3: Edición del BeanInfo

Es importante que escribamos perfectamente el nombre de la clase, con su paquete incluido, y terminando en .class. Si no lo hacemos así, no dará ningún error pero el editor personalizado no aparecerá.

Otro detalle importante es que el BeanInfo debe ser regenerado si cambiamos la definición de nuestro componente (si añadimos o quitamos propiedades o métodos).

Paso 6 – Compilar y agregar el componente a la paleta

Después de los pasos anteriores, solo queda recompilar el proyecto (Clean and build) y agregar el componente a la paleta. Si agregamos el componente a un proyecto de prueba, veremos como nuestra propiedad puede ser editada con nuestro editor personalizado (ver Ilustración 4).

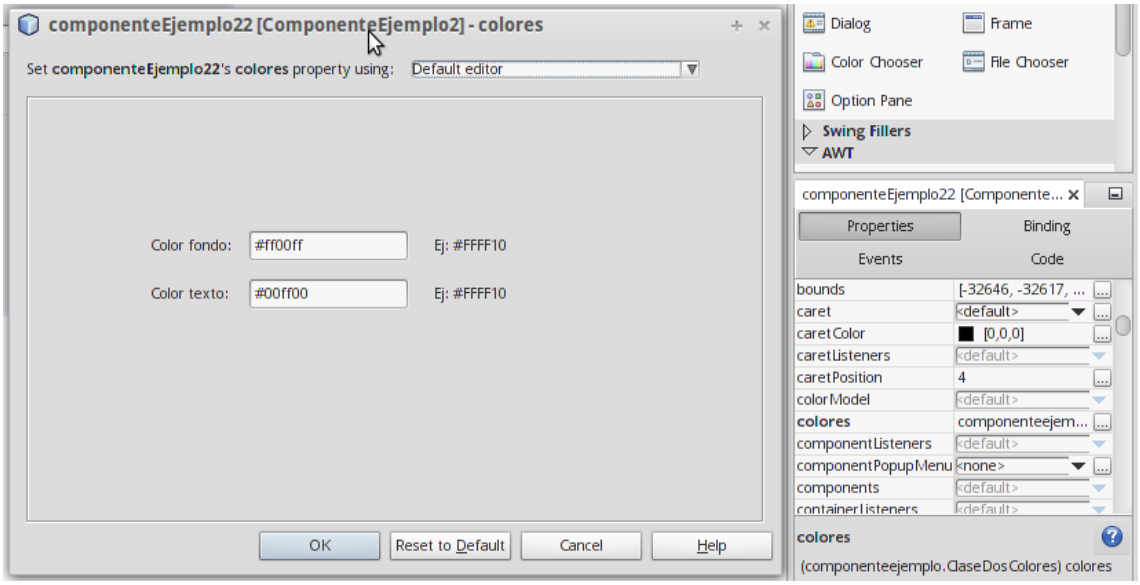

Ilustración 4: Editor personalizado en funcionamiento

Con esto finalizo la explicación de como hacer un editor de propiedades personalizado. Faltaría agregar la funcionalidad al componente (podéis verlo acabado en el código adjunto). No la incluyo en el documento porque no aporta nada nuevo a lo que ya he explicado para el primer ejemplo.