CREACIÓN DE COMPONENTES

Contenido

Creación de componentes (Beans)

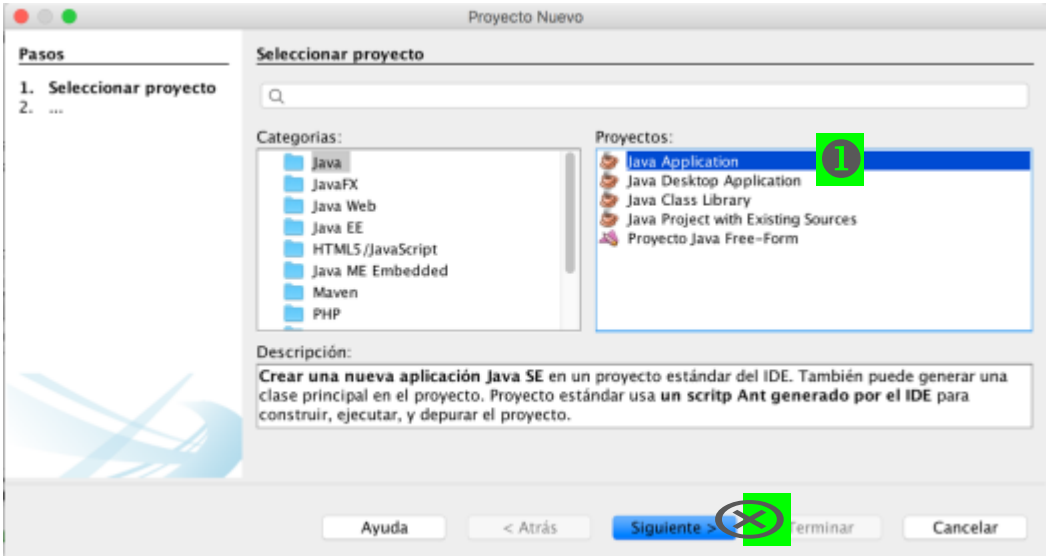
Para crear un componente (Bean) podemos realizarlo de las siguientes formas:

1. Crear un JPanel con los componentes que conforman el Bean.
2. Crear una clase que herede del componente a personalizar y editar sus propiedades.
3. Crear una clase que herede del componente a personalizar y un editor de propiedades personalizado.

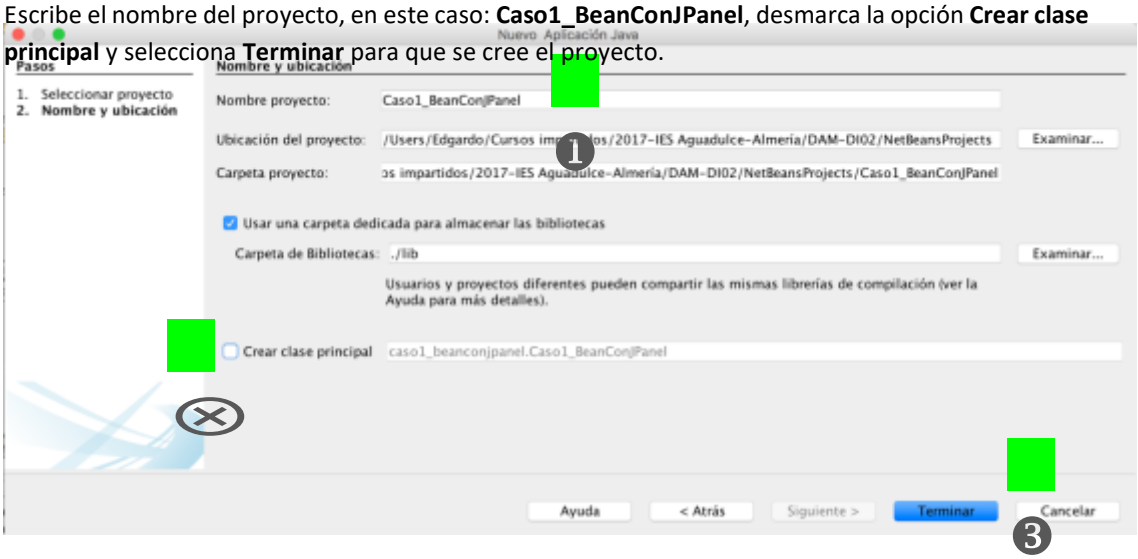
A continuación, se muestra un ejemplo de cada caso.

Caso 1: Creación de Bean con JPanel

1. Crea un nuevo proyecto: **Archivo → Proyecto Nuevo…**
2. Elige la opción **Java Application** y selecciona **Siguiente**

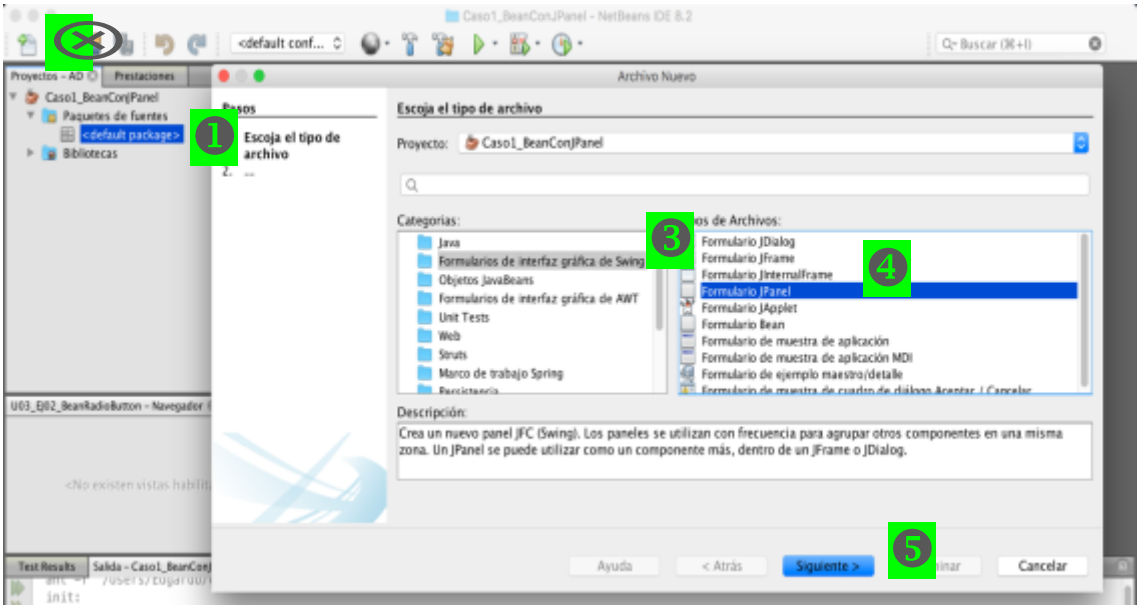


3. Escribe el nombre del proyecto, en este caso: **Caso1_BeanConJPanel**, desmarca la opción **Crear clase principal** y selecciona **Terminar** para que se cree el proyecto.

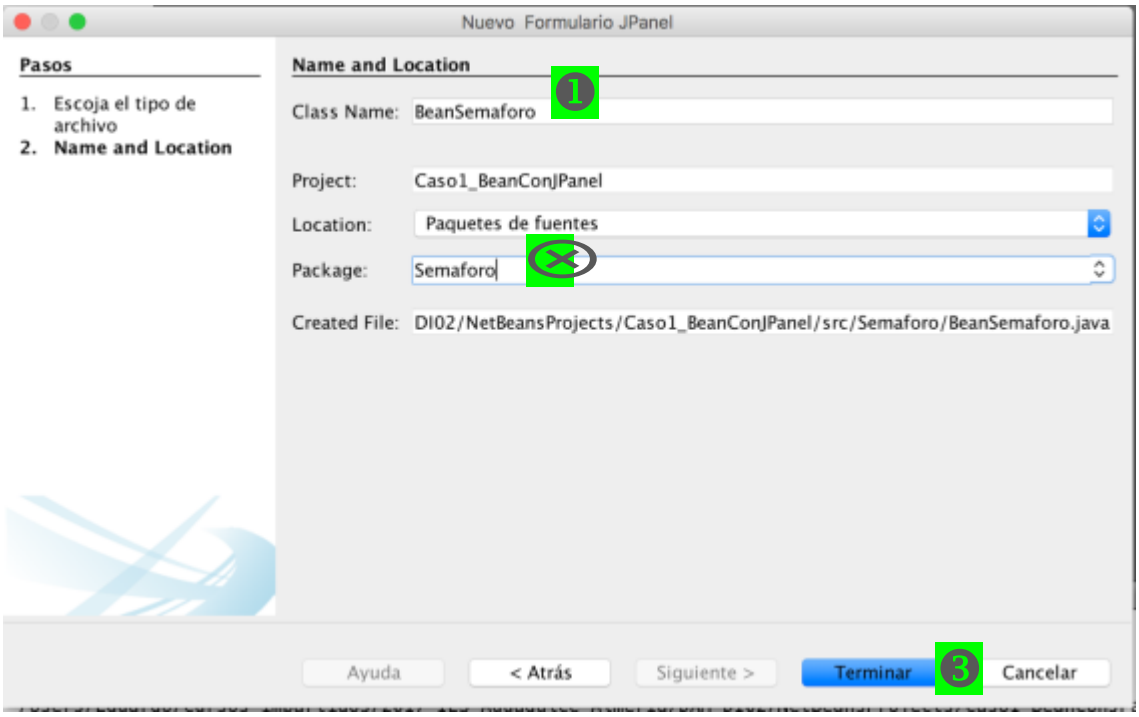


4. Ahora crea un JPanel que será el componente personalizado.
   Selecciona las siguientes opciones:

   1. <Default package>
   2. Archivo Nuevo
   3. Formulario de interfaz gráfica de Swing
   4. Formulario JPanel
   5. Siguiente



5. Escribe el nombre del JPanel, en este caso se llamará **Semaforo**, el nombre del package (**BeanSemaforo**) y pulsa **Terminar**



6. Luego realiza el siguiente diseño con los siguientes componentes:
    1. **JLabel**: Semáforo (Componente)
    2. **ButtonGroup**: para agrupar los JRadioButton
    3. **JRadioButton**: Rojo, Amarillo, Verde
    4. **JPanel**: para visualizar el color seleccionado

7. Una vez creado el diseño debes programar la lógica para el componente, para ello utiliza los eventos para cada opción a seleccionar:

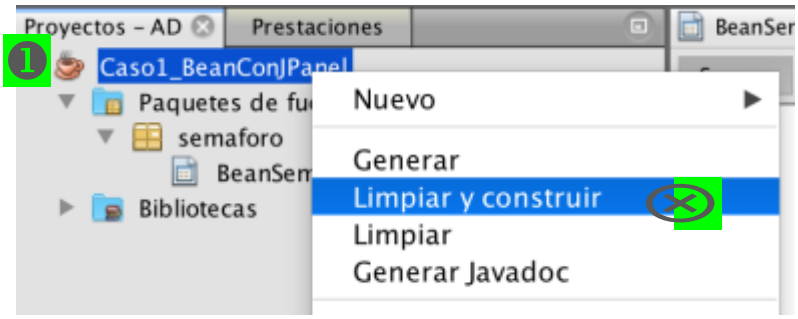
```
private void rbRojoActionPerformed(java.awt.event.ActionEvent evt) {
        jpSemaforo.setBackground(Color.RED);
    }

    private void rbAmarilloActionPerformed(java.awt.event.ActionEvent evt) {
        jpSemaforo.setBackground(Color.YELLOW);
    }

    private void rbVerdeActionPerformed(java.awt.event.ActionEvent evt) {
        jpSemaforo.setBackground(Color.GREEN);
    }
}
```

También agrega los getters y setters para poder obtener y configurar al componente. A modo ejemplo utiliza solamente los gettters y setters de los JRadioButton

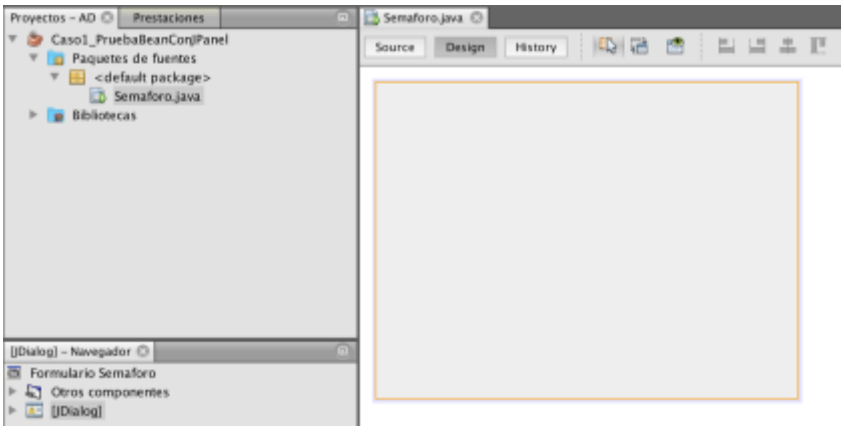
```
//GETTERS Y SETTERS
    public ButtonGroup getBgSemaforo() {
        return bgSemaforo;
    }

    public void setBgSemaforo(ButtonGroup bgSemaforo) {
        this.bgSemaforo = bgSemaforo;
    }

    public JRadioButton getRbAmarillo() {
        System.out.println("rbAmarillo");
        return rbAmarillo;
    }
```

```
public void setRbAmarillo(JRadioButton rbAmarillo) {
    this.rbAmarillo = rbAmarillo;
}

public JRadioButton getRbRojo() {
    return rbRojo;
}

public void setRbRojo(JRadioButton rbRojo) {
    this.rbRojo = rbRojo;
}

public JRadioButton getRbVerde() {
    return rbVerde;
}

public void setRbVerde(JRadioButton rbVerde) {
    this.rbVerde = rbVerde;
}
```

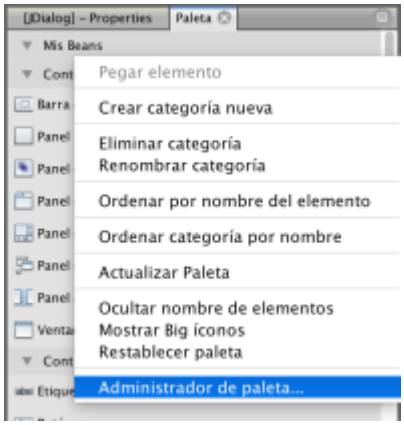
8. Cuando esté finalizado hay que generar el componente, para ello:
    1. Selecciona el proyecto y despliega el menú contextual
    2. Elige la opción **Limpiar y contruir** para generar el archivo *.jar*



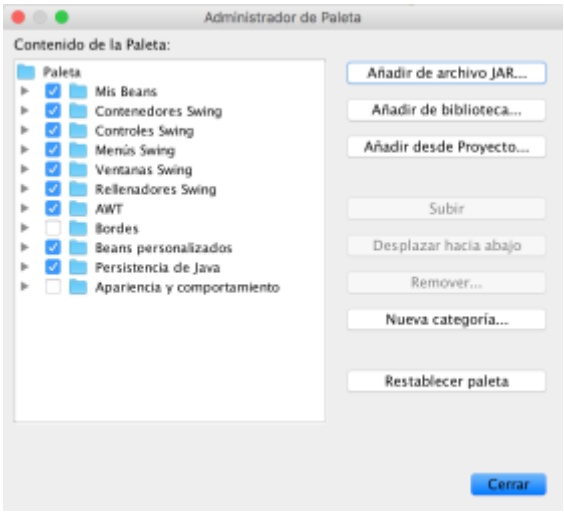
9. Ahora es posible cargar el componente a la paleta. Para ello primero crea un nuevo proyecto, en este caso he utilizado el nombre **Caso1_PruebaBeanConJPanel**.
   Dentro de él crea un formulario JDialog, en este ejemplo lo he creado con el nombre **Semaforo**



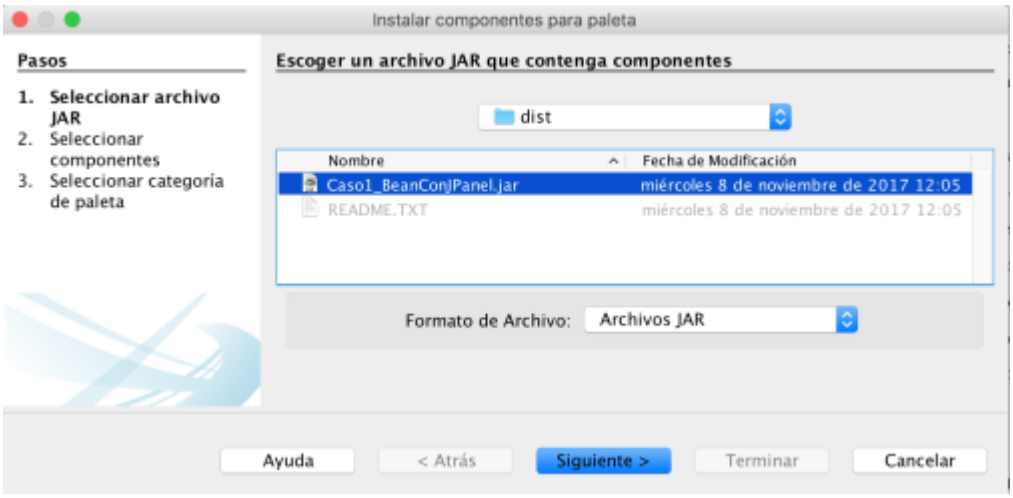
10. Ahora es el momento de agregar el componente a la paleta para poder utilizarlo. Para ello:
    - Despliega el menú contextual de la paleta y selecciona la opción **Administrador de paleta…**



    - Selecciona **Añadir de archivo JAR…**

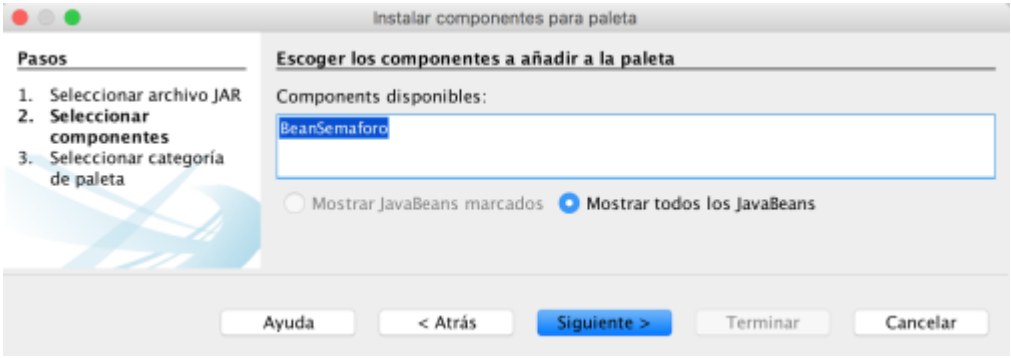


    - Selecciona el componente creado con la compilación realizada en el punto **8** y pulsa en el botón **Siguiente**. El componente se encuentra dentro del directorio **dist** ubicado en el directorio del proyecto. En este caso: ***Caso1_BeanConJPanel/dist***

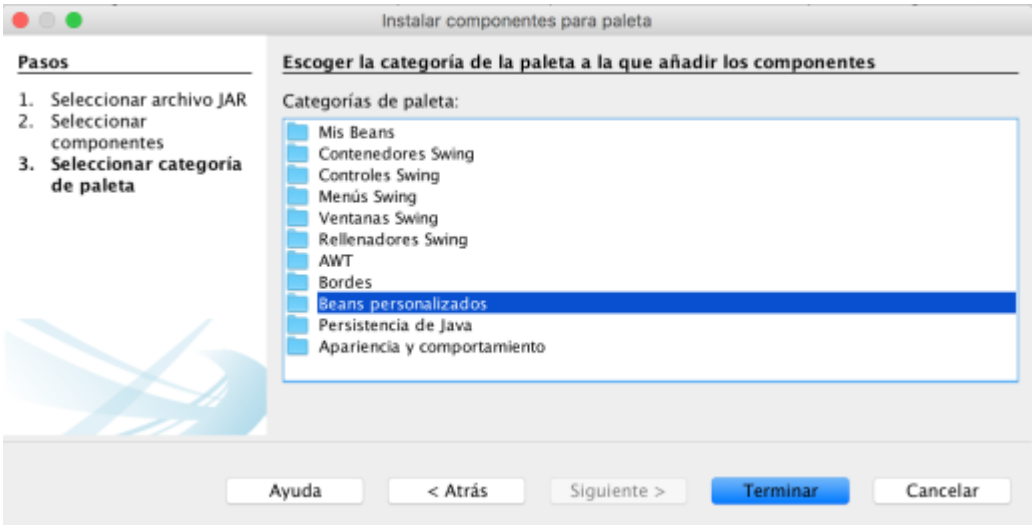


Creación de componentes

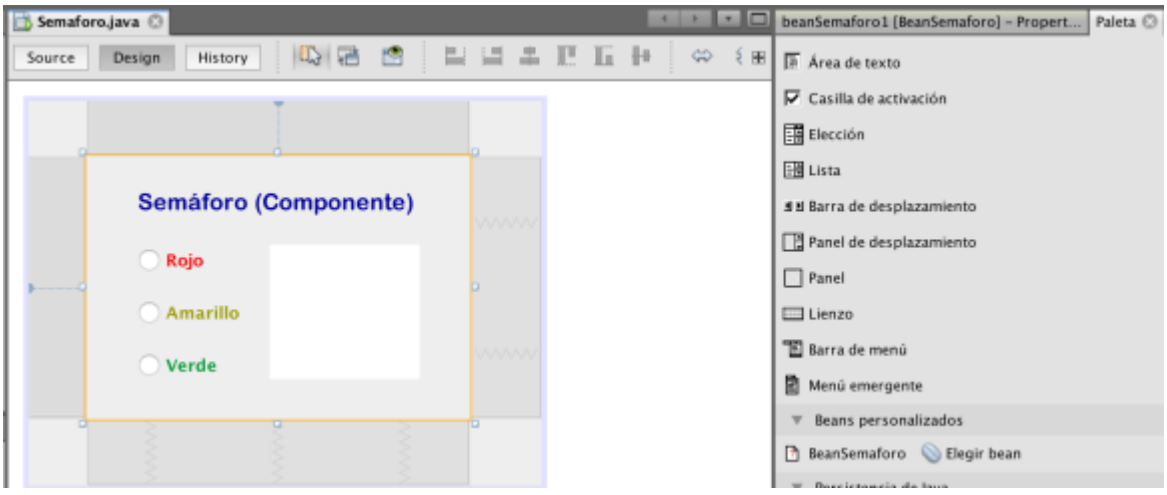
- Selecciona el componente **BeanSemaforo** y pulsa en el botón **Siguiente**



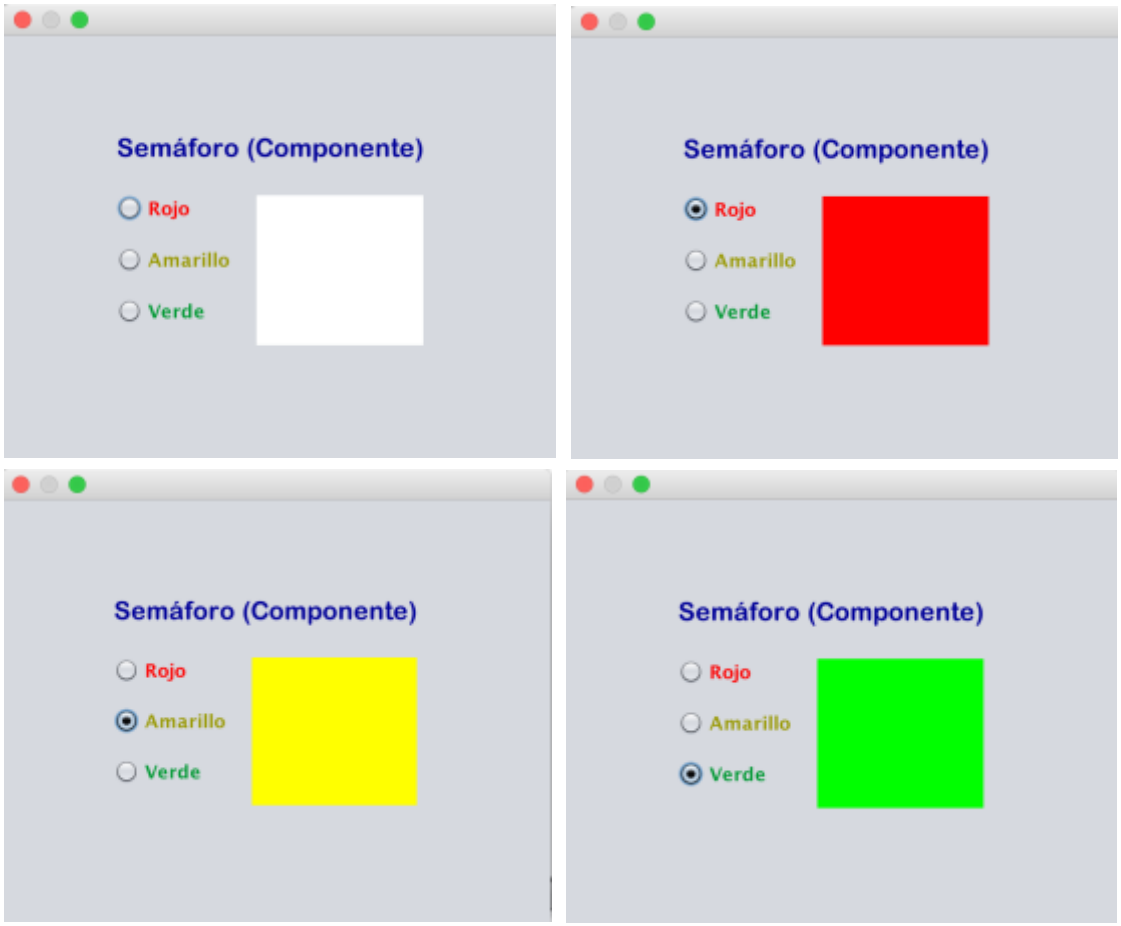
- Selecciona la categoría en la que desees agregar el componente. Es conveniente agregarlo en una categoría diferente a las ya establecidas por NetBeans, por ejemplo en **BeansPersonalizados**.



- Ahora agrega el componente al JDialog arrastrándolo como cualquier otro componente.



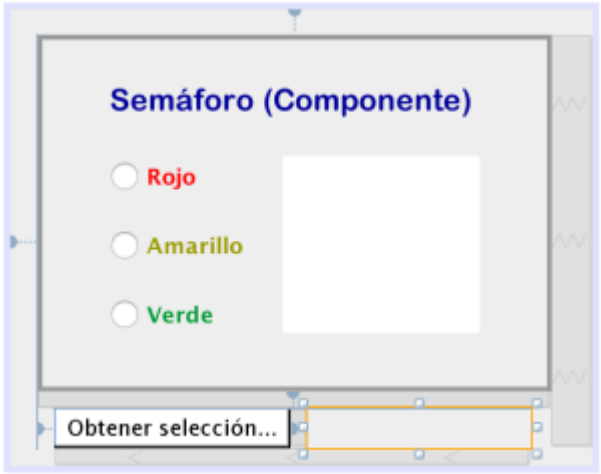
- Verifica su funcionamiento ejecutando el proyecto.



Interactuando con el Bean

Ahora incorpora la funcionalidad a tu programa para interactuar con el Bean insertado. Para ello añade un JButton para obtener el nombre del JRadioButton seleccionado y un JLabel en el que se visualizará el valor obtenido. ¡Vamos a ello!

1. Modifica la interfaz para que quede de la siguiente forma:



2. Los nombres utilizados para los controles son los siguientes:

   - beanSemaforo: **beanSemaforo1**
   - JButton: **btnObtenerSeleccion**
   - JLabel: **lblDatos**

3. Añade la funcionalidad al botón **Obtener selección…**
   Este procedimiento obtiene todos los elementos (JRadioButton) del JButtonGroup del componente, los recorre e imprime el valor de aquel que esté seleccionado.

```
private void btnObtenerSeleccionActionPerformed(java.awt.event.ActionEvent
evt) {
        Enumeration e = beanSemaforo1.getBgSemaforo().getElements();

        while (e.hasMoreElements()==true){
            JRadioButton r=(JRadioButton) e.nextElement();
            if (r.isSelected()){
                lblDatos.setText(r.getText());
            }
        }
}
```

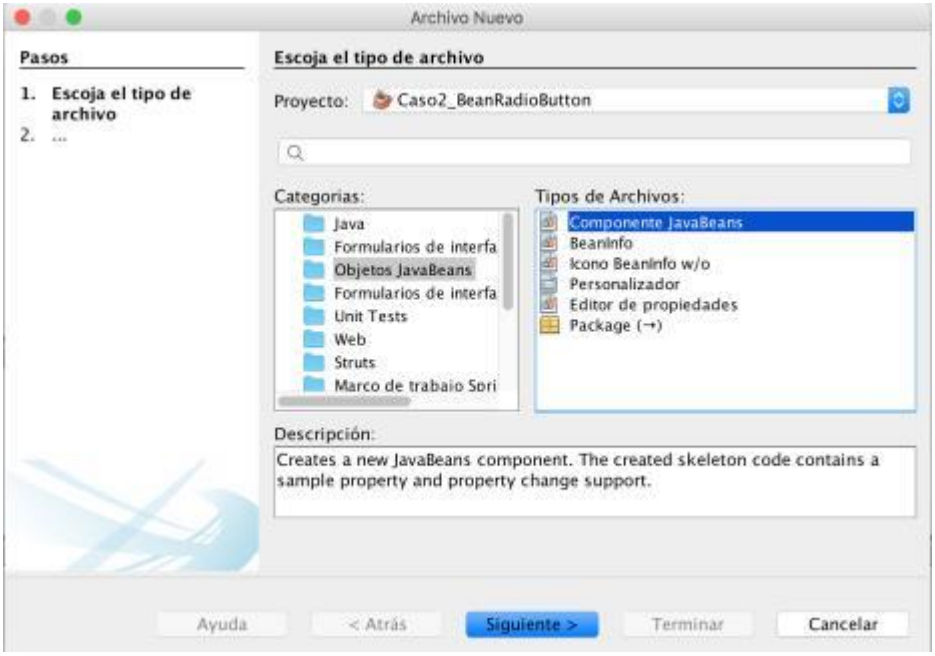
4. Verifica el funcionamiento de tu aplicación.

Caso 2: Creación de Bean y edición de propiedades

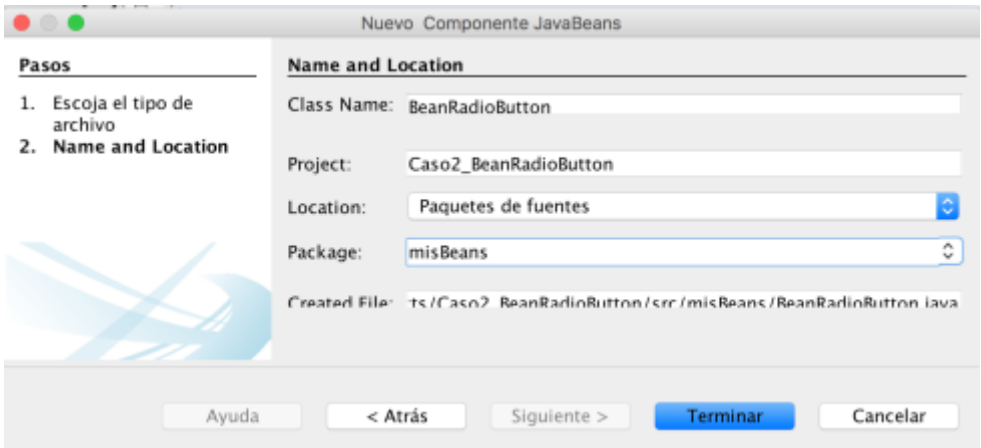
Para crear un componente personalizado se puede partir desde uno que ya exista y luego personalizar sus propiedades. A continuación, se describirán las acciones que debes realizar y considerando que algunas de ellas son similares no se pondrán imágenes en esos apartados.

Creación del componente

1. Crea un nuevo proyecto con el nombre **Caso2_BeanRadioButton**
2. Agrega un componente JavaBeans. Selecciona las siguientes opciones:
   - <Default package>
   - Archivo → Nuevo
   - Objetos JavaBeans
   - Componente JavaBeans
   - Siguiente



- Como nombre de clase escribe **BeanRadioButton** y en Package escribe **misBeans** y selecciona **Terminar**



3. A continuación, elimina todo el código de la clase dejando solamente el **constructor**. Además, debes **extender** la clase al componente que vas a implementar, en este caso **JRadioButton**.

```
public class BeanRadioButton extends JRadioButton implements Serializable {
    public BeanRadioButton() {
    }
}
```

Agregando propiedades al bean

1. Agrega los atributos que consideres necesarios a la nueva clase e inicialízalos en el contructor. Luego crea los setters y getters los cuales puedes personalizar.

```
public final class Bean_RadioButton extends JRadioButton implements
Serializable { String
    texto; Color
    colorTexto;
    boolean habilitado;
    boolean invisible;
    boolean seleccionado;
    Color colorSeleccion;

    public Bean_RadioButton() {
        this.texto="Item1";
        this.colorTexto=Color.BLUE;
        this.habilitado=true;
        this.invisible=false;
        this.seleccionado=false;

        this.setTexto(this.texto);
        this.setColorTexto(this.colorTexto);
        this.setHabilitado(this.habilitado);
        this.setInvisible(this.invisible);
        this.setSeleccionado(this.seleccionado);
    }

    /***********************************************/
    /* GETTES Y SETTERS                            */
    /***********************************************/
    public String getTexto() {
        return "El texto de la etiqueta es: " + texto;
    }

    public void setTexto(String texto) {
        this.texto = texto;
        this.setText(this.texto);
    }

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

```
    public void setColorTexto(Color colorTexto) {
        this.colorTexto = colorTexto;
        this.setForeground(this.colorTexto);
    }

    public boolean isHabilitado() {
        return habilitado;
    }

    public void setHabilitado(boolean habilitado) {
        this.habilitado = habilitado;
        this.setEnabled(this.habilitado);
        if (this.habilitado) System.out.println("Componente activado");
        if (!this.habilitado) System.out.println("Componente deactivado");
    }

    public boolean isInvisible() {
        return invisible;
    }

    public void setInvisible(boolean invisible) {
        this.invisible = invisible;
        this.setVisible(!this.invisible);
    }

    public boolean isSeleccionado() {
        return seleccionado;
    }

    public void setSeleccionado(boolean seleccionado) {
        this.seleccionado = seleccionado;
        this.setSelected(this.seleccionado);
    }
}
```

Puedes observar que, de esta forma, aprovechando la introspección es posible crear todo tipo de métodos para nuestros componentes. Además, es posible redefinir dentro de la clase del vean los métodos propios del componente del cual hereda.
Observa la función **setHabilitado**, además de habilitar el componente muestra un mensaje por consola.

Creación del proyecto

Aquí crearemos el proyecto en el cual insertaremos el Bean que acabamos de crear. Además, agregaremos algunos botones para utilizar los métodos definidos en el componente. Por último, programaremos la interfaz que será implementada en el componente para manejar el evento del clic sobre JRadioButton.
Crea una interfaz como la de la imagen siguiente.

CÓDIGO DEL PROYECTO

Seguidamente, se puede ver el código necesario que se implementa en cada evento de botón. A cada uno se le asigna una de las funciones definidas en el Bean.

- En el hilo de ejecución, a modo didáctico, se inicializa el componente y se utilizan las funciones definidas en él.

```
public void run() {
   new PruebaRadioButton().setVisible(true);
   rb1.setTexto("Negro");
   rb1.setColorTexto(Color.BLACK);
   lblMensaje.setText(rb1.getTexto());

   rb1.setSeleccionado(true);
   if (rb1.isSeleccionado())
       System.out.println("El elemento está seleccionado");

   rb1.setSeleccionado(false);
   if (!rb1.isSeleccionado())
      System.out.println("El elemento no está seleccionado");

   rb1.setEnabled(false);
   rb1.setEnabled(true);
}
```

- Los eventos de los botones se programan así:

```
private void btnColorNegroActionPerformed(java.awt.event.ActionEvent evt) {
    rb1.setColorTexto(Color.BLACK);
    rb1.setTexto("Negro");
    lblMensaje.setText(rb1.getTexto());
}

private void btnColorRojoActionPerformed(java.awt.event.ActionEvent evt) {
    rb1.setColorTexto(Color.RED);
    rb1.setTexto("Rojo");
    lblMensaje.setText(rb1.getTexto());
}

private void btnMarcarActionPerformed(java.awt.event.ActionEvent evt) {
    rb1.setSeleccionado(true);
    lblMensaje.setText("Has marcado: " + rb1.getText());
}

private void btnDesmarcarActionPerformed(java.awt.event.ActionEvent evt) {
    rb1.setSeleccionado(false);
    lblMensaje.setText("Has desmarcado: " + rb1.getText());
}

private void btnHabilitarActionPerformed(java.awt.event.ActionEvent evt) {
    rb1.setEnabled(true);
    lblMensaje.setText("Componente habilitado");
}

private void btnDeshabilitarActionPerformed(java.awt.event.ActionEvent evt) {
    rb1.setEnabled(false);
    lblMensaje.setText("Componente deshabilitado");
}
```

El componente resuelve un evento

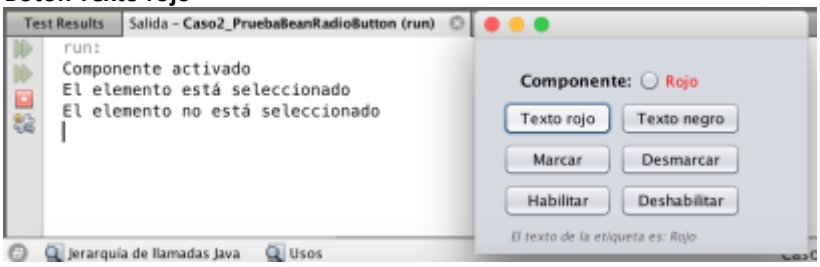
```
public BeanRadioButtonYo() {
      this.texto="Item";
      this.colorTexto=Color.BLUE;
      this.habilitado=true;
      this.invisible=false;
      this.seleccionado=false;

      this.setTexto(this.texto);
      this.setColorTexto(this.colorTexto);
      this.setHabilitado(this.habilitado);
      this.setInvisible(this.invisible);
      this.setSeleccionado(this.seleccionado);

      //Listener con clase anonima
      this.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            System.out.println("Marcado");
            JOptionPane.showMessageDialog(null, "Marcado");
          }
      });
   }
```

Prueba del proyecto

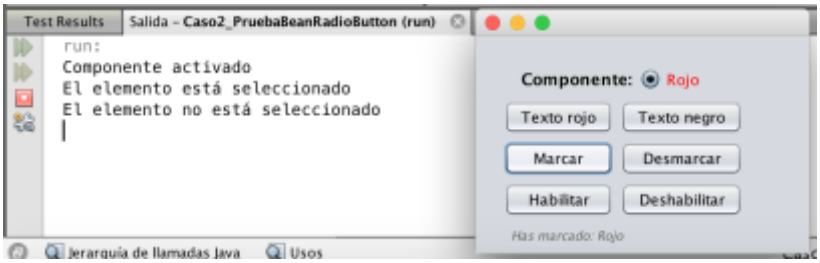
Ahora es el momento de probar la funcionalidad del proyecto. Debemos probar los eventos incluidos en el proyecto como los del componente (El implementado en la interfaz)

PROBANDO LOS BOTONES

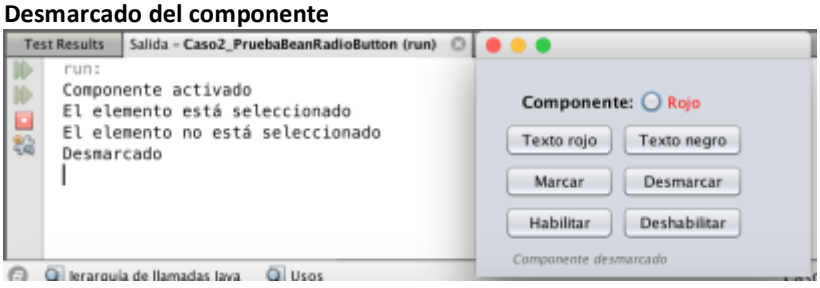
Botón Texto rojo



Botón Marcar



Desmarcado del componente



Caso 2A: Creación de Bean y edición de propiedades. Asociación de Acciones a Eventos.

Ejemplo "Temporizador"
Tema T3 página 30.
8. Elaboración de un componente de ejemplo.
Para la elaboración de un componente, debes tener muy claros los pasos que debes dar.

1. Creación del componente.
2. 2. Adición de propiedades.
3. Implementación de su comportamiento.
4. Gestión de los eventos.
5. Uso de componentes ya creadas en NetBeans.

En este caso se añada a lo estudiado en los ejemplos anteriores el apartado 4. Gestión de Eventos.
Los componentes Java utilizan el modelo de delegación de eventos para gestionar la comunicación entre objetos.

Caso 3: Creación de Bean con PropertyEditor

Comenzaremos en este caso a crear un componente de texto y una interfaz como se muestra a continuación:

Componente de texto (Clase ComponenteTexto.java)

```
package MisBeans;

import java.awt.Color;
import java.awt.Font;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.io.Serializable;
import javax.swing.JTextField;

/**
 *
 * @author Edgardo Capallo
 */
public class ComponenteTexto extends JTextField implements Serializable {

    private int ancho;      //Cantidad de caracteres
    private String tipo;    //Tipo de variable
    private Color color;    //Color de texto
    private Font fuente;    //Fuente de texto

    //Constructor (Establece los valores por defecto al instanciar la clase ComponenteTexto
    public ComponenteTexto() {
        this.ancho=5;
        this.tipo="Texto";
        this.color=Color.BLACK;
        this.fuente=new Font("Agency FB", Font.BOLD, 14);
        this.setText("");
        this.gestionaEntrada(); //Gestiona la entrada por teclado
    }

    /**
     * Obtiene el color
     * @return devuelve el valor correspondiente al color
     */
    public Color getColor() {
        return color;
    }

    /**
     * Asigna el valor a color
     * @param color es el nuevo valor a asignar
     */
    public void setColor(Color color) {
        this.color = color;
        this.setForeground(color);
    }

    /**
     * Devuelve el valor del ancho
     * @return devuelve el valor de ancho
     */
    public int getAncho() {
        return ancho;
    }
```

```
/**
 * Asigna el valor al ancho
 * @param ancho es el nuevo valor a asignar
 */
public void setAncho(int ancho) {
    this.ancho = ancho;
    super.setColumns(ancho);
}

/**
 * devuelve el valor de fuente
 * @return devuelve el valor de fuente
 */
public Font getFuente() {
    return fuente;
}

/**
 * Asigna el valor a fuente
 * @param fuente nuevo valor a asignar
 */
public void setFuente(Font fuente) {
    this.fuente = fuente;
    this.setFont(fuente);
}

/**
 * Obtiene el tipo
 * @return devuelve el tipo
 */
public String getTipo() {
    return tipo;
}

/**
 * Asigna el tipo
 * @param tipo nuevo valor a asignar
 */
public void setTipo(String tipo) {
    if ( tipo.equals("Entero") || tipo.equals("Texto"))
          this.tipo = tipo;
    }

//Redefine el método setText para interpretar las nuevas variables (Entero y SN)
@Override
public void setText(String text) {
    switch (tipo) {
        case "Entero":
            try{
                Integer.parseInt(text);
                super.setText(text);
            }
            catch (NumberFormatException e){
                super.setText("");
            }
            break;
        default:
            super.setText(text); //asume el texto definido en las propiedades del componente
            break;
    }
}
```

```
    @Override
    public String getText() {
        return super.getText();
    }

    /**
     * Gestiona la entrada por teclado de los distintos tipos
     */
    public final void gestionaEntrada() {
        this.addKeyListener(new KeyAdapter() {

            @Override
            public void keyTyped(KeyEvent e) {
                char caracter = e.getKeyChar();

                switch (tipo) {
                    case "Entero":
                        if (!(Character.isDigit(caracter)
                            ||(getText().length()>=ancho)) {
                            e.consume();
                        }
                        break;


                    case "Texto":
                        //Limita el ingreso al ancho establecido
                        if (getText().length()>=ancho) e.consume();

                        break;
                }
            }
        });
    }
}
```

Interfaz (Clase PruebaComponente.java)

Código dentro de la interfaz

```
public class PruebaComponente extends javax.swing.JFrame {

  /**
   * Creates new form PruebaComponente
   */
  public PruebaComponente() {
     initComponents();
     ctNick.setTipo("Texto");
     ctNick.setAncho(12);

     ctEdad.setTipo("Entero");
     ctEdad.setAncho(2);
     ctEdad.setText("Esto no es un entero"); //Para que muestre el error

     lblComentarios.setEditable(false);
     lblComentarios.setText(lblComentarios.getText()+
          "(1) - Tipo: " + ctNick.getTipo() +
          " - Ancho: por defecto: " + ctNick.getAncho()+"\n");
     lblComentarios.setText(lblComentarios.getText()+
          "(2) - Tipo: " + ctEdad.getTipo() +
          " - Ancho: " + ctEdad.getAncho()+"\n");
  }
```

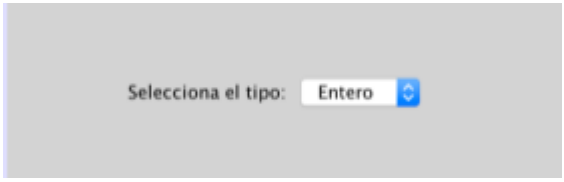
Hasta aquí es similar a los casos anteriores, se define un componente de texto, sus propiedades y sus métodos y algunas funciones adicionales que nos permiten tratar los nuevos tipos de datos definidos para este componente (Entero y Texto).

A continuación, se definirá un editor de propiedades que permitirá establecer determinadas propiedades del componente desde la hoja de propiedades.

Creando el panel del editor (Clase EditorTipoPanel.java)

Composición

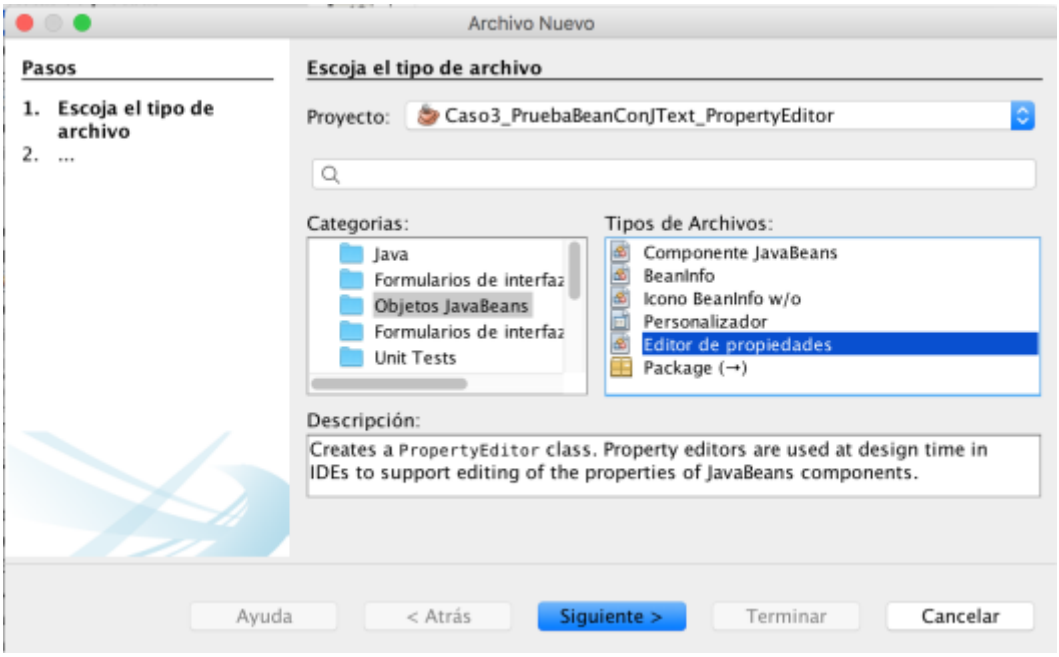
- La clase es un JPanel
- JComboBox:
  - Nombre de variable: **cbTipo**
  - Valores: Entero, SN



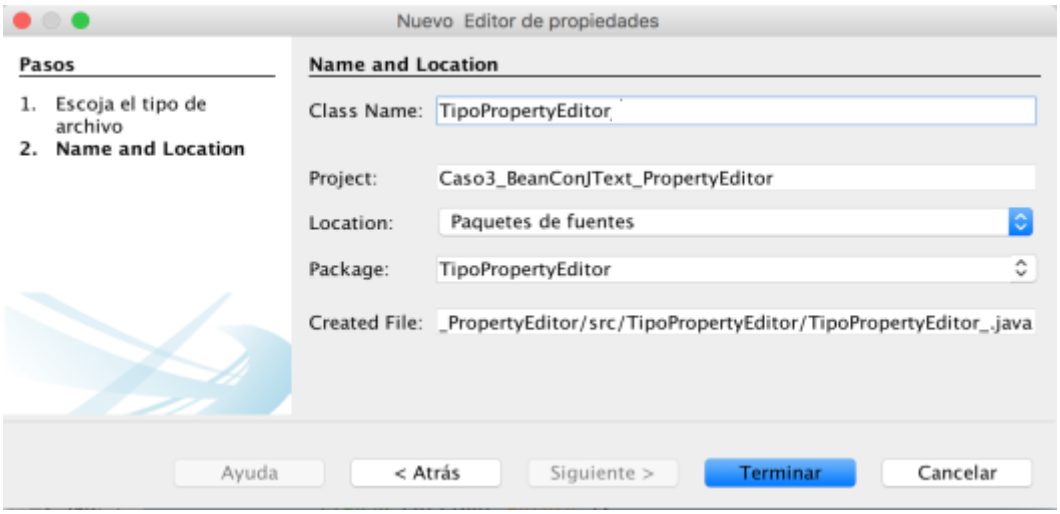
Creando el editor de propiedades (Clase TipoPropertyEditor.java)

Aquí es donde se implementa la interfaz y se definen los métodos para configurar la propiedad desde la hoja de propiedades.

- Crea un nuevo archivo dentro del paquete misBeans



- Asigna el nombre a la clase (En este caso: **TipoPropertyEditor**)



Creación de componentes

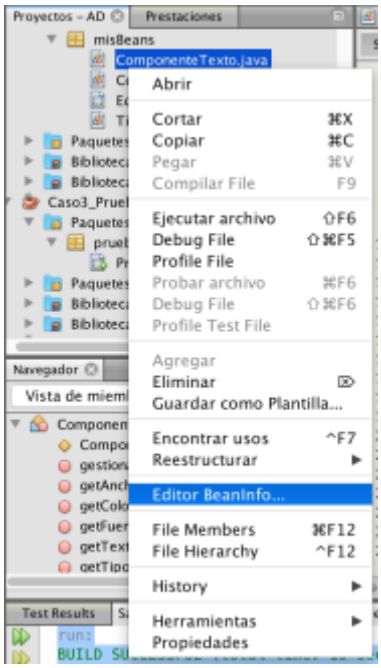
- Programa el editor

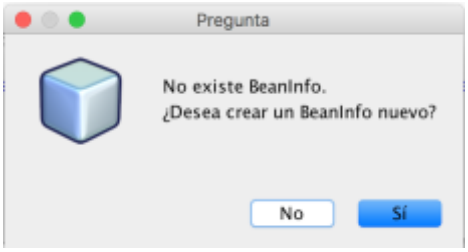
```
public class TipoPropertyEditor extends PropertyEditorSupport {
    //su único atributo es un EditorTipoPanel
    private EditorTipoPanel editor=null;

    public TipoPropertyEditor() {
        this.editor=new EditorTipoPanel();
    }

    //Para indicar que existe un editor personalizado
    @Override
    public boolean supportsCustomEditor() {
        return true;
    }

    //Obtiene el editor personalizado
    @Override
    public Component getCustomEditor() {
        return editor;
    }

    //El constructor de beans solicita el valor del objeto para almacenar la
    //propiedad, llamando al método getValue () .
    //Obtiene el valor del Editor y lo devuelve a la hoja de propiedades
    @Override
    public Object getValue() {
        if (super.getValue()==null){
            setValue(null);
        }
        String ret=(String)super.getValue();
        ret=editor.cbTipo.getSelectedItem().toString();

        return ret;
    }

     //El constructor de beans le dice al editor de propiedades el valor actual
     //de la propiedad llamando al método setValue ()
    //Asigna el valor en la hoja de propiedades
    @Override
    public void setValue(Object value) {
        if (value==null){
            value=new String();
        }
        super.setValue(value);
    }

    //Permite definir la lista en la hoja de propiedades, establece los valores
    //posible para la propiedad en la hoja de propiedades.
    @Override
    public String[] getTags() {
        String[] tags={"Entero", "Texto"};
        return tags;
    }

//Cuando el usuario cambia el valor de Cadena , el constructor de bean le dice a
//la propiedad el nuevo valor llamando al método setAsText () .
//Asigna el valor definido en la hoja de propiedades a través del getTags o escrito a mano
//Además asigna como item seleccionado en el editor al valor seleccionado del tag
//establece el valor seleccionado en el editor de propiedades en el editor
@Override
    public void setAsText(String text){
        editor.cbTipo.setSelectedItem(text);
        super.setAsText(text);
    }

//Devuelve en la hoja de propiedades el valor establecido en el Editor
//Devuelve el valor de la propiedad como texto

@Override
    public String getAsText(){
            String ret = (String) super.getAsText();
            ret=editor.cbTipo.getSelectedItem().toString();
            return ret;
```

```
        }

    ///El constructor de beans solicita el código que inicializará la propiedad
    //llamando a getJavaInitializationString ()
    //Define como se tratará el elemento seleccionado en el editor.
    //Sobreescritura del método que devuelve el valor del componente en la clase
    //correspondiente al editor de propiedades del componente (tipoPropertyEditor):
    /*La función ?getJavaInitializationString() tiene que devolver sí o sí una cadena.
    Java, luego elimina las comillas dobles de la cadena y coloca el valor devuelto en el código.
    Por ejemplo, en el componente de texto se envía la cadena "[cadena_de_texto]"
    porque el componente requiere una cadena para el tipo.*/
    //idem
    /*@Override
        public String getJavaInitializationString() {
                 return "\"" + editor.cmbTipo.getSelectedItem() + "\"";
        }*/

          @Override
          public String getJavaInitializationString() {
              StringBuilder sb = new StringBuilder();
              sb.append("\"");
              sb.append((String) editor.cbTipo.getSelectedItem());
              sb.append("\"");
              String retorno = new String(sb);
              return retorno;
          }
      }
```

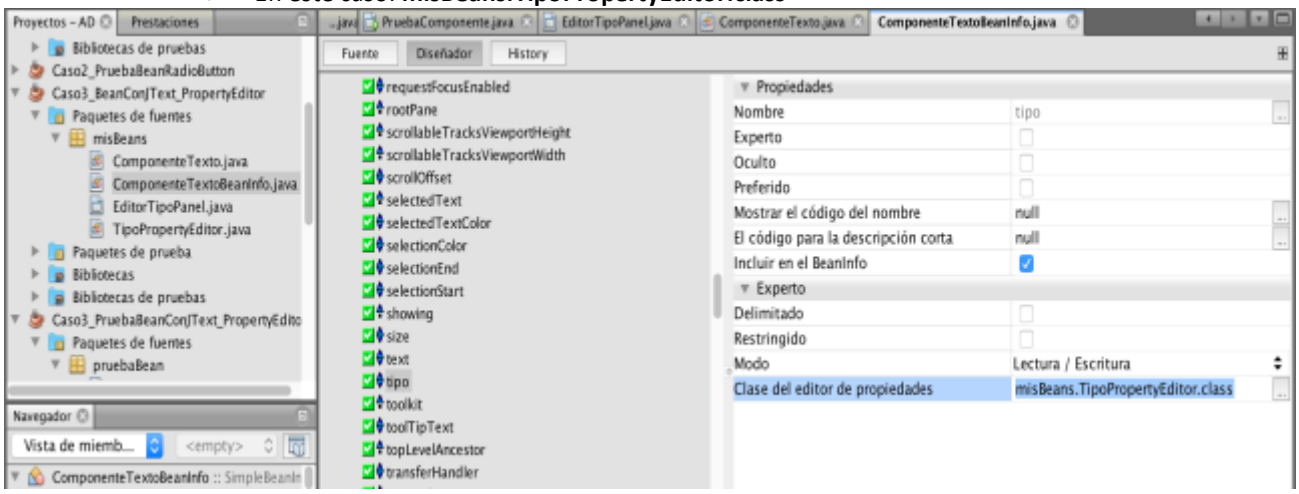
Asociando el editor de propiedades (Clase ComponenteTextoBeanInfo.java)

- Cuando selecciones **Editor BeanInfo…** te preguntará si deseas crear el BeanInfo. Selecciona **Sí**.





- Una vez creado el BeanInfo ábrelo con el **Diseñador**
  - Busca la propiedad **tipo**
  - Escribe la clase del editor de propiedades incluyendo el paquete en el que se encuentra:
    - En este caso: **misBeans.TipoPropertyEditor.class**

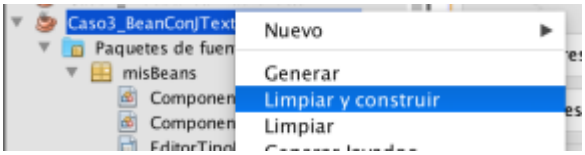


- No olvides guardar los cambios.

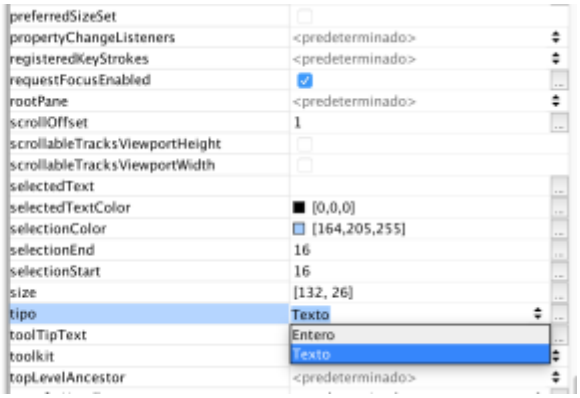
Ahora es necesario limpiar y construir el bean y volver a insertar los componentes en el proyecto de prueba.

Probando el proyecto

- Limpia y recontruye el componente



- Elimina los componentes que habías insertado anteriormente y vuelve a cargar los nuevos componentes. No olvides renombrarlos con los mismos nombres.
- Fíjate que al seleccionar un componente, en hoja de propiedades aparece la propiedad que hemos creado (**tipo**)



- Dentro de esa propiedad es posible seleccionar entre los valores establecidos (Entero y Texto) gracias a que se ha redefinido el método **getTags()**
- Además, si seleccionas **…** puede abrir el editor de propiedades que has creado.