Pasos para crear un componente.

- Crear una clase java que exitienda de un componente de la paleta

PEJ:

public class RelojDigital extends JLabel implements Serializable

{

public RelojDigital()

{

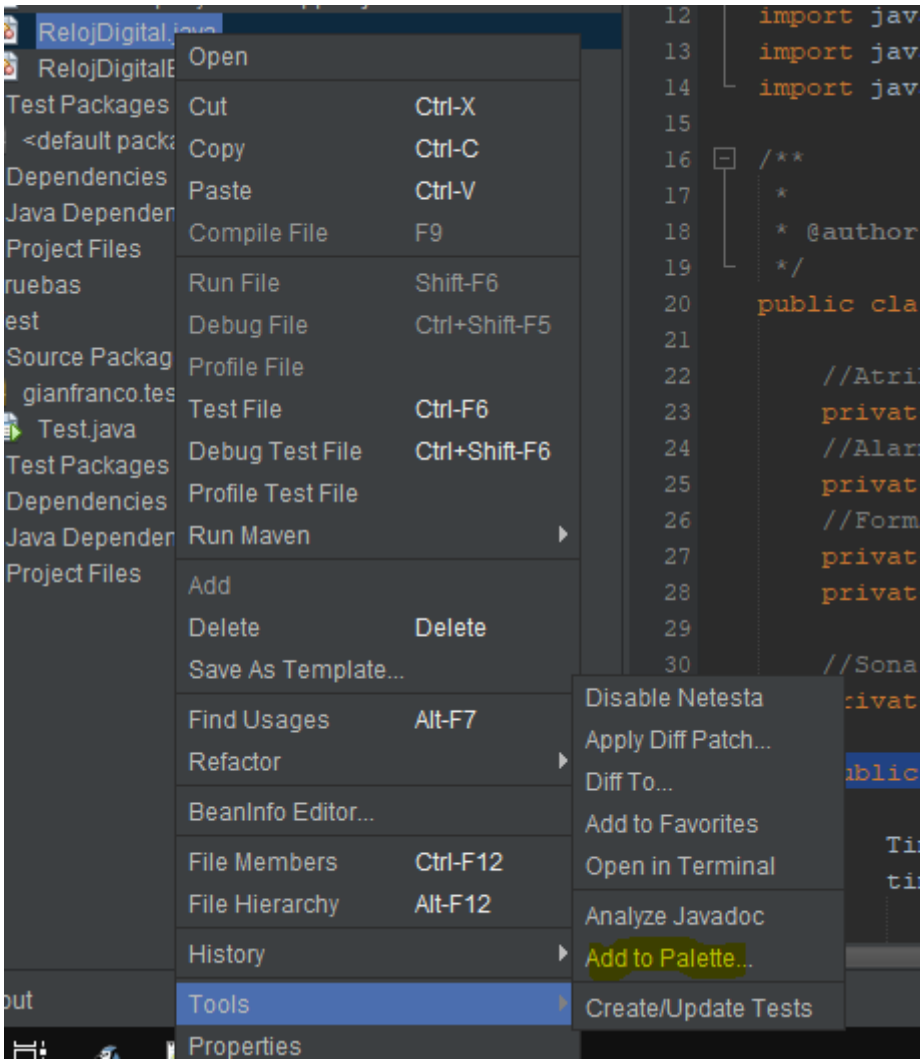
//implementar un constructor sin parámetros

}

Implementar getters y setters de atributos generados por netbeans.(importante respetar el nombre)

}

- Ya tendriamos el componente que Podemos añadir a la palleta:



- Si el componente tiene que tener un atributo que sea objeto con propiedades específicas debemos crear el objeto en una nueva clase para definer sus atributos:

En este caso el componente tiene un objeto alarma:

Creamos la clase Alarma- importante que sea serializable!

PEJ:

```
public class Alarma implements Serializable
{
        Añadir getters y setters
}
```

- Si queremos pasar los datos de la propiedad en este caso alarma desde un jform,

  Tendremos que crear el jform qu obtendra los datos:

  -crear una nueva clase jpanel con la sopciones que se quiera y modificar el codigo

  -crear el método que recojera los datos del panel

  PEJ:

```
/**
     * Metodo para crear la alarma a partir de lo valores introducidos en el panel
     * @return
     */
    public Alarma getSelectedValue()//llamarlo siempre asi
    {
        Date date = (Date)jSpinnerAlarma.getValue();
        boolean activa = jCheckBoxActiva.isSelected();
        return new Alarma(date,activa);//Esto sera lo que  se devuelva cuando se setee la propiedad
    }
```

- Para que netbeans sepa que la propiedad alarma de un componente esta personalizado es decir en este caso se cogen los datos de un jframe creado por nostros

  Hay que decirselo, esto se hace creando una nueva clase que extienda de PropertyEditorSupport

PEJ:

Hay que redefinir estos 4 métodos como mínimo para que los datos introducidos en la propiedad cuando se ejecute la aplicacion se enlacen, si no se hace esto el componente solo funciona en modo diseño.

*/

public class AlarmaPropertyEditorSupport extends PropertyEditorSupport

{

//Crear un panel para pasarlo

private AlarmaPanel alarmaPanel = new AlarmaPanel();

@Override//Este meétodo hace que netbeans sepa que hay propiedades definidas de forma distinta a la predefinida

public boolean supportsCustomEditor() {

return true; //True = si hay propiedades definidas que no serán las de"fabrica"

}

@Override//Este métdo pasa el jpanel que se pasa al editar la propiedad

public Component getCustomEditor() {

return alarmaPanel;//Pasar el panel para elegir la alarma.

}

@Override//Este método devuelve un String “adecuado” es decir que pueda inciar el componente con la propiedad definida de manera adecuada en ejecución de la aplicación, de no hacerlo solo funcionaría en modo diseño

//al ser String hay que “indicar todo” si se quiere acceder a una clase hay que “importarla” entera en el string con el paquete entero definido y los tring que puedan crear objetos si es el caso

public String getJavaInitializationString() {

//Tiene que devolver un string a partir del cual se creara el objeto

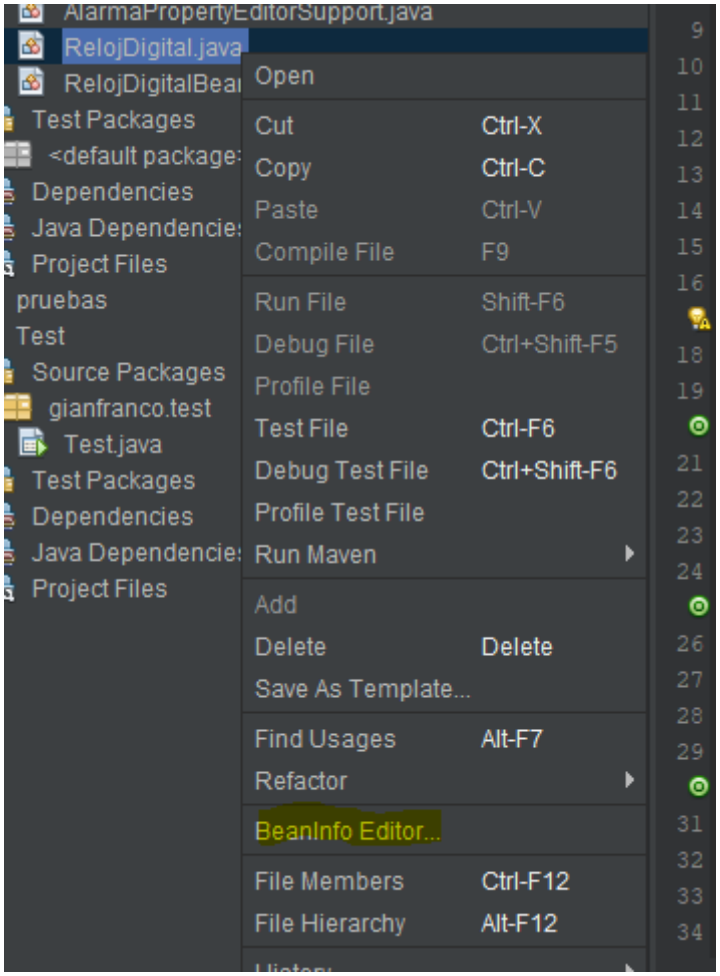
return "new gianfranco.relojDigital.Alarma(new java.util.Date(" + alarmaPanel.getSelectedValue().getHoraAlarma().getTime() + "l)," + alarmaPanel.getSelectedValue().isActiva() + ")"; //Devovler el valor que tendra setAlarma para que funcione en ejecución

@Override//Esto es el valor va a tener la propiedad tipo object, tomará el valor de los datos generados por el jpanel que se devuelven mediant eel método selectedValue() definido en el jpanel

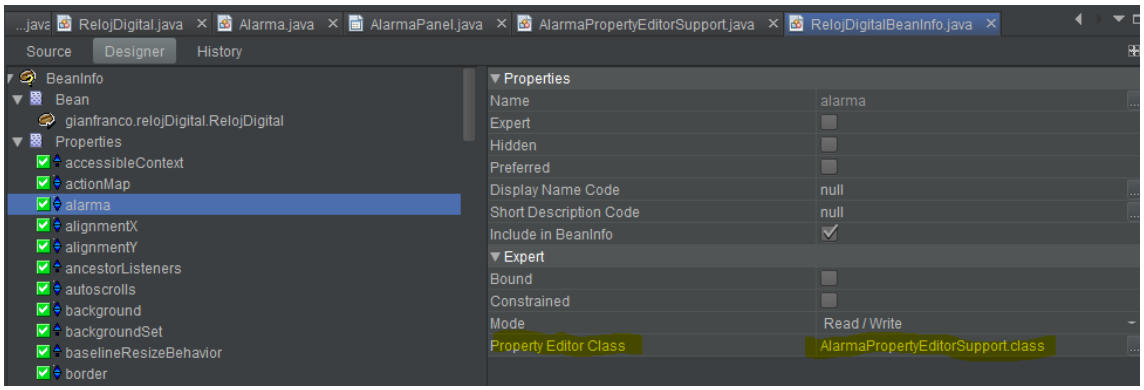
```
    public Object getValue() {
        return alarmaPanel.getSelectedValue(); //Devolver el valor del panel
    }


}
```

- Hay que genear el jbean.info



Se generara una clase en la que hay que indicar a la propiedad que hemos creado que se enlace con la clase PropertyEditorSupport que hemos creado:




Importante el .class

Con esto el componente ya estará definido.

Clean and build para actulizar el componente si ya ha sido añadido a la paleta.

- Para añadir una funcionalidad

Crear una clase listener, un iterfaz:

```
public interface AlarmaListener
 {
   public void sonarAlarma();
}
```

- En la clase se define la funcionalidad que se recibirá:

Un método add para pasarle el método de la interfaz.

```
/**
    * Lo que hace la alarma al sonar
    *
    * @param alarmaListener
    */
   public void addAlarmaListener(AlarmaListener alarmaListener)
   {
      this.alarmaListener = alarmaListener;
   }
```

Donde se quiera que se haga ese algo llamar al metodo PEJ:

```
alarmaListener.sonarAlarma();
```

- En el Código donde se usa el componente habrá que redefinir el método de la interfaz para pasarle la funcionalidad:

Al obejto relojDigital1 creado a partir del componente se le pasa un objeto nuevo de la interzaf implementado el método de la interfaz con su funcionabilidad

```
relojDigital1.addAlarmaListener(new AlarmaListener()
        {
           @Override
           public void sonarAlarma()
           {
              JOptionPane.showMessageDialog(Test.this,"Espabila!!!!");
           }
        });
```

La idea es crear una interfaz que tenga el método de la funcionabilidad, no implementada ya que es un interfaz.

La clase del componente tiene qu ser capaz de crear “la interfaz”;esto es el listner no sabe lo que hacer pero recibe codigo que le dice lo que hará cuando se implement eel método de la interfaz.

Se crea un objeto de la interfaz.

```
private AlarmaListener alarmaListener;
```

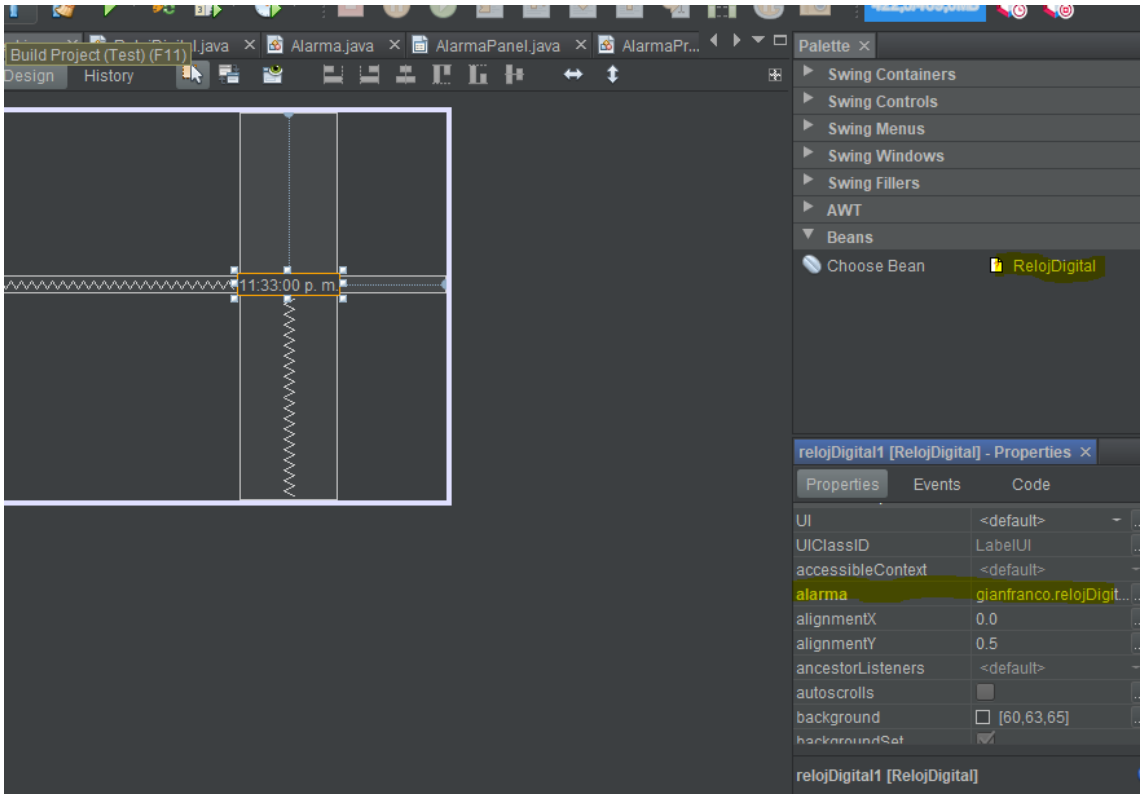
Se crear un método para “pasar un objeto Interfaz” , ya que es un interfaz cuando se pase hay que sobreescribir el método de la interfaz obligatoriamente.

```
public void addAlarmaListener(AlarmaListener alarmaListener)
 {
     this.alarmaListener = alarmaListener;
  }
```

En el código donde se usa el componente:

```
relojDigital1.addAlarmaListener(new AlarmaListener()
        {
            @Override
            public void sonarAlarma()
            {
                JOptionPane.showMessageDialog(Test.this,"Espabila!!!!");//Esto es lo que realemte se hace
            }
        });
```

Componente y propiedad:




Si no funciona el añadir un nuevo componente—clean and build