Fase 1. Definición de la clase principal (clase que contendrá el componente).

Para empezar, crearemos un nuevo proyecto (Java Aplication) sin la clase Main.

Dentro de dicho proyecto crearemos un paquete (Java package) , donde guardaremos todas nuestra clases.

Ya dentro del paquete crearemos la clase principal que contendrá el componente. Esta clase va a heredar de JLabel y se implementa como Serializable.

```
public class Planificador extends JLabel implements Serializable, ActionListener {
```

A continuación, se debe añadir a la clase las propiedades:

-formatoHora booleana. Para indicar si es 24 ó 12h,

```
public boolean isFormatoHora() {
    return formatoHora;
}

public void setFormatoHora(boolean formatoHora) {
    this.formatoHora = formatoHora;
    actualizarEtiqueta();   // Actualiza la etiqueta de la alarma con los datos de la propiedad Alarma.
}
```

-formatoFecha booleana. Para indicar si es formato dd/MM/aaaa ó dd/MM/aa.

```
public boolean isFormatoFecha() {
    return formatoFecha;
}

public void setFormatoFecha(boolean formatoFecha) {
    this.formatoFecha = formatoFecha;
    actualizarEtiqueta();   // Actualiza la etiqueta de la alarma con los datos de la propiedad Alarma.
}
```

Tendréis que incluir nuevas variables del tipo Timer y otra del tipo LocalDateTime

Aquí tendréis que implementar varios métodos, dentro del constructor, necesitáis crear un método que establezca un temporizador para actualizar la fecha y hora cada segundo.

```
public void setAlarma(Alarma alarma) {

    if(t != null){
        setActivo(false);   // Si ya existe una alarma programada se detiene el temporizador.
    }

    this.alarma = alarma;
    actualizarEtiqueta();   // Actualiza la etiqueta de la alarma con los datos de la propiedad Alarma.

    LocalDateTime fechaActual = LocalDateTime.now();
    LocalDateTime fechaAlarma = this.alarma.getFechaHora();
    long diferenciaTiempo = ChronoUnit.MILLIS.between(fechaActual, fechaAlarma);
Se comprueba la diferencia entre la fecha actual y la que se ha programado para la alarma.
    t = new Timer((int)diferenciaTiempo, this);
    // Temporizador que se dispara cuando pasa el tiempo entre la fecha actual y la programada.
    setActivo(true);    // Activar el temporizador.
}
```

También método para comprobar la fecha y hora actuales y la fecha de una alarma definida.

- Alarma, dentro del paquete crearemos una clase **nueva.** Dicha clase se implementará como **SERIALIZABLE** y contendrá **dos atributos fecha/hora y tarea)**. Para dicha clase, **se implementará el constructor y los getters y setters para sus dos atributos.**

También deberemos colocar el constructor público de la clase sin parámetros.

```
public class Alarma implements Serializable{

    private LocalDateTime fechaHora;
    private String tarea;


    public Alarma(LocalDateTime fechaHora, String tarea) {
        this.fechaHora = fechaHora;
        this.tarea = tarea;
    }

    public LocalDateTime getFechaHora() {
        return fechaHora;
    }

    public void setFechaHora(LocalDateTime fechaHora) {
        this.fechaHora = fechaHora;
    }

    public String getTarea() {
        return tarea;
    }

    public void setTarea(String tarea) {
        this.tarea = tarea;
    }


}
```

Fase 2. Creación de un editor de propiedades personalizado.

En esta fase creareis el editor personalizado. Añadir al paquete de nuestro proyecto un JPanel Form.

Aquí tendréis que incorporar un método público que devuelva el valor de la propiedad seleccionada en el panel, es decir un objeto del tipo editor de propiedades. Devolverá una cadena que representa la fecha y hora del jSpinner y el texto obtenido del jTextField.

```
    // Se extraen los datos de los dos componentes y se devuelven para ser gestionados por el PropertyEditor.
    public Alarma getSelectedValue(){

        LocalDateTime fechaHora = ((Date)jSpinnerFechaHora.getValue())
                .toInstant()
                .atZone(ZoneId.systemDefault())
                .toLocalDateTime();
// Se extrae la fecha del componente en formato Date y se convierte a LocalDateTime.
        String tarea = jTextFieldTarea.getText();   // Se extrae el texto del componente.

        return new Alarma(fechaHora, tarea);    // Se devuelve un objeto Alarma con los datos extraídos.
    }
```

No olvidar que en tarea a realizar tendréis que establecer restricción de String de longitud máxima de 30 caracteres.

```
    private void jTextFieldTareaKeyTyped(java.awt.event.KeyEvent evt) {
        if(jTextFieldTarea.getText().length() == 30){
            evt.consume();  // Si se escribe mas de 30 caracteres se genera e
        }
    }
```

Este método es necesario porque no se puede acceder al panel desde la clase PropertyEditor por ser una propiedad privada de la clase.

Fase 3. Creación de la clase PropertyEditor para relacionar el editor con la propiedad.

Añadir una clase "PropertyEditor" , que hereda de PropertyEditorSupport y su constructor.

```
public class AlarmaPropertyEditor extends PropertyEditorSupport{

    PanelAlarma panelAlarma = new PanelAlarma();
```

Definir 4 de los métodos de la clase "PropertyEditorSupport", para ello pulsar botón derecho raton "Insert Code" "Override Method" y marcar los métodos (getCustomEditor,supportCustomEditor,getValue,getJavaInitializationString)

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.

```
    public boolean supportsCustomEditor() {
        return true;
    }
```

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

```
public Component getCustomEditor() {
        return panelAlarma;
```

3.- **Método getValue**.- Este método devolvería el resultado de la llamada al método getSelectedValue del objeto **panel** que hemos declarado como propiedad de la clase. Recordemos que ese método getSelectedValue lo implementamos anteriormente para devolver el valor seleccionado por el usuario en el panel de la propiedad personalizada.

```
  public Object getValue() {

        return panelAlarma.getSelectedValue();
    }
```

4.- **Método getJavaInitializationString**.- El método constaría sólo de la siguiente línea:

return String.format("\"%s\"", nuestropanel.getSetValue());

Devolvería la cadena que representa el valor de nuestropanel.getSetValue() entre comillas dobles.

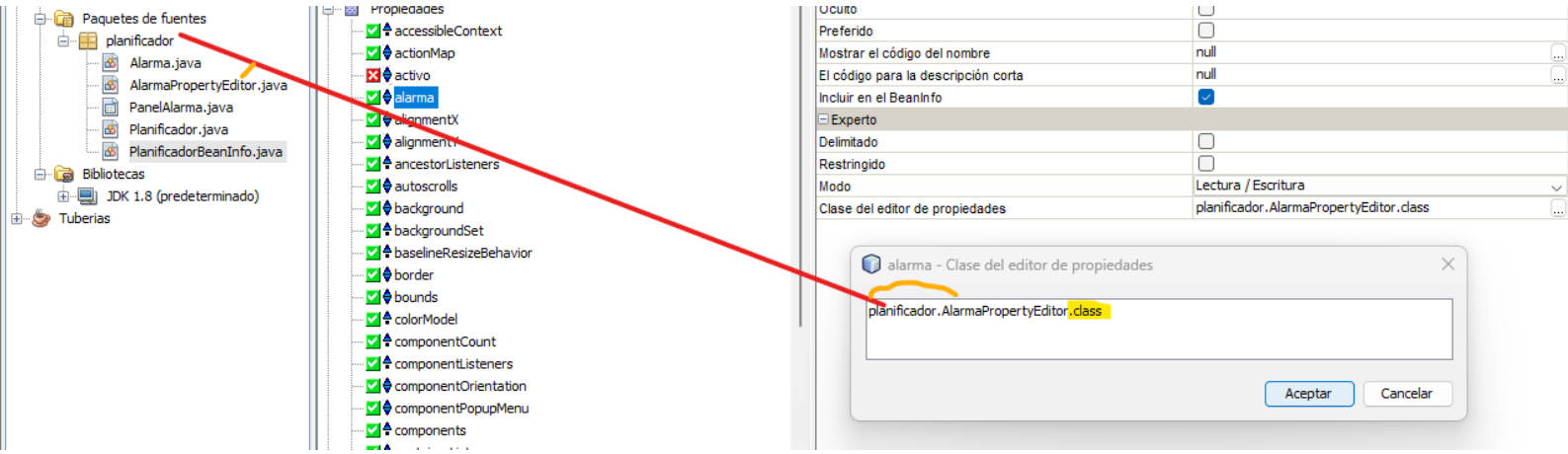
```
  public String getJavaInitializationString() {

       // Se extrae la fecha del Jpanel PanelAlarma.
        LocalDateTime fechaHora = panelAlarma.getSelectedValue().getFechaHora();
         // Se extrae el texto de la tarea del Jpanel PanelAlarma.
        String tarea = panelAlarma.getSelectedValue().getTarea();
      return "new Alarma(LocalDateTime.parse(\"" + fechaHora.toString() + "\"), \"" +
tarea + "\")";      }
```

Fase 4. Creación del BeanInfo para conectar la clase que hereda de PropertyEditorSupport a la propiedad.

Crear el BeanInfo que relaciona la clase basada en PropertyEditorSupport con la propiedad para permitirnos usar el editor personalizado.

Para crear el BeanInfo, es pulsar boton derecho del ratón sobre la clase principal con el botón derecho del ratón y elegir "BeanInfo Editor..." . Acceder a la vista de diseño de la clase y seleccionar nuestra propiedad **alarma**.

En la parte derecha, en la propiedad "PropertyEditor Class" , debeis escribir la clase editor de propiedades incluyendo el paquete en donde está definida .class.



Fase 5. Incorporación de eventos y listener para detectar la alarma programada.

Se programa una fecha y hora que debe generar un evento de aviso cuando llegue a esa fecha y hora. El avisó debe informar de la tarea a realizar. Debe estar activo hasta que el usuario confirme la lectura del mismo. Una vez leído ese aviso se podrá volver a programar un nuevo aviso de alarma.

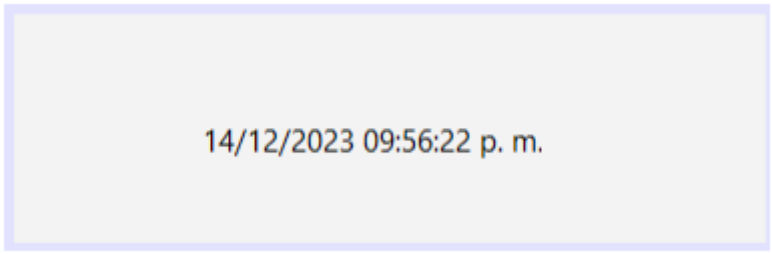
```
    @Override
    public void actionPerformed(ActionEvent e) {

        t.stop();   // Cuando se acaba el tiempo del temporizador se detiene.
        listener.capturarActivarAlarma(new AlarmaEvent(this));  // Genera el evento de la alarma.

    }

    // Clase de tipo evento (EventObject) que servirá para generar el evento.
    public class AlarmaEvent extends EventObject{
        public AlarmaEvent(Object source){
            super(source);
        }
    }

    // Interfaz para añadir los oyentes a otras clases.
    public interface ActivarAlarma extends EventListener{
        void capturarActivarAlarma(AlarmaEvent evt);
    }

    // Se registra un oyente.
    public void addActivarAlarma(ActivarAlarma listener){
        this.listener = listener;
    }

    // Se elimina un oyente.
    public void removeActivarAlarma(ActivarAlarma listener){
        this.listener = null;
    }

}
```

FASE 6.- Obtener el componente, agregarlo a la paleta y crear un proyecto para comprobar su funcionamiento

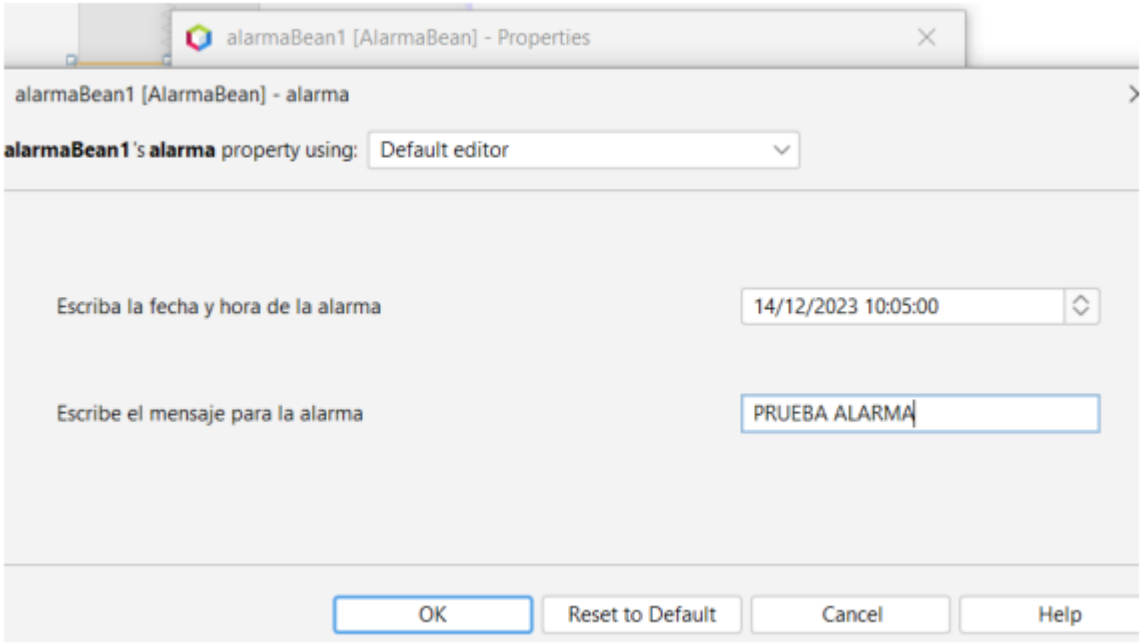
Llegados a este punto, se puede **ejecutar "Clean and Build"** o "Limpiar y Construir" sobre la clase termometro.java y agregarlo a la categoría de la paleta de componentes que consideremos oportuna.

Una vez agregado a la paleta, **crearemos una "Java Application"** y probaremos el componente creado.

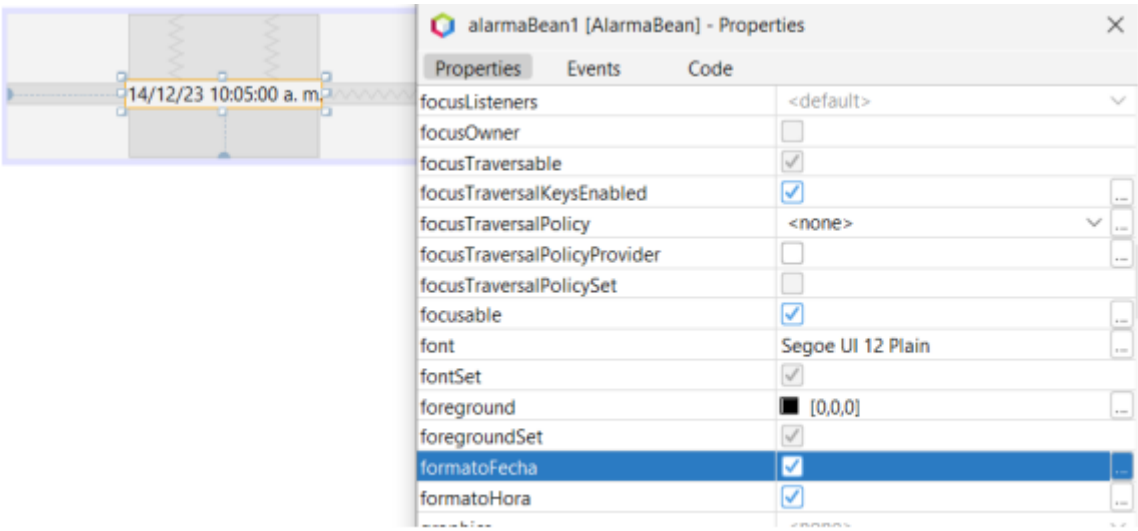
Imagen con el componente insertado tomando hora y fecha sistema, por defecto toma formato fecha larga y formato 12 hora :



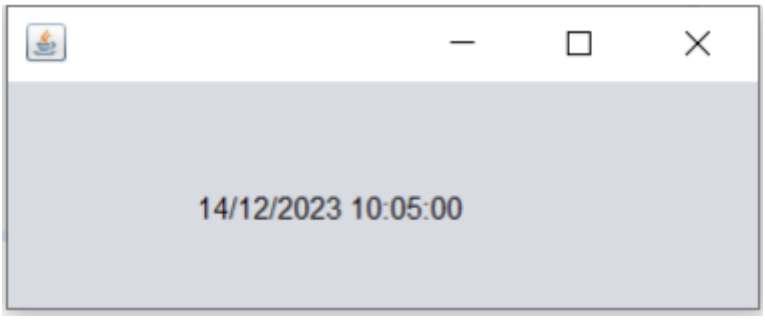
Configuramos nuestra alarma desde las propiedades de nuestro componente, para que salte a las 10:05:00



En mi caso el formato de fecha y hora lo cambio también desde las propiedades, pero se puede hacer de varias formas incluso que se cambie desde el formulario de prueba.



Quedando así, es decir nos mostrara la fecha y hora a la que está programada la alerta.



A la hora indicada 10:05:00 mostrara el mensaje programado.