

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

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

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

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

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

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

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.

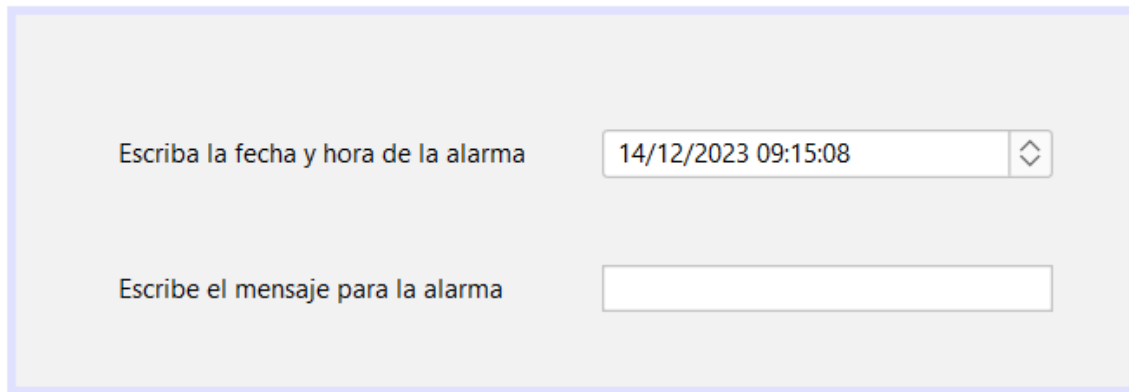
También método para comprobar la fecha y hora actuales y la fecha de una alarma definida.

- 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.

## 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.



Escriba la fecha y hora de la alarma 14/12/2023 09:15:08

Escribe el mensaje para la alarma

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 `TextField`.

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

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.

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

A continuación, os recuerdo 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 `panel` que hemos declarado como propiedad de la clase.

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 (true o false según el usuario haya elegido en el panel "Celsius" o "Fahrenheit").

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.

#### **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 botón 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.**

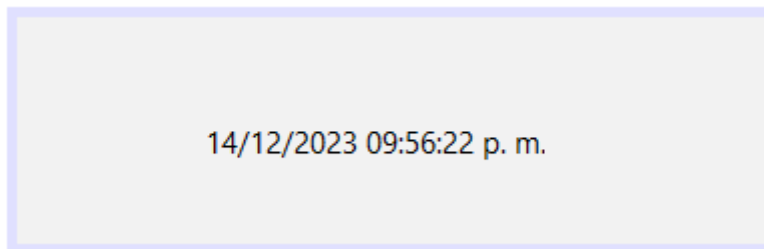
Se programa una fecha y hora que debe generar un evento de aviso cuando llegue a esa fecha y hora. El avisó debe informar de la tarea a realizar. Debe estar activo hasta que el usuario confirme la lectura del mismo. Una vez leído ese aviso se podrá volver a programar un nuevo aviso de alarma.

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

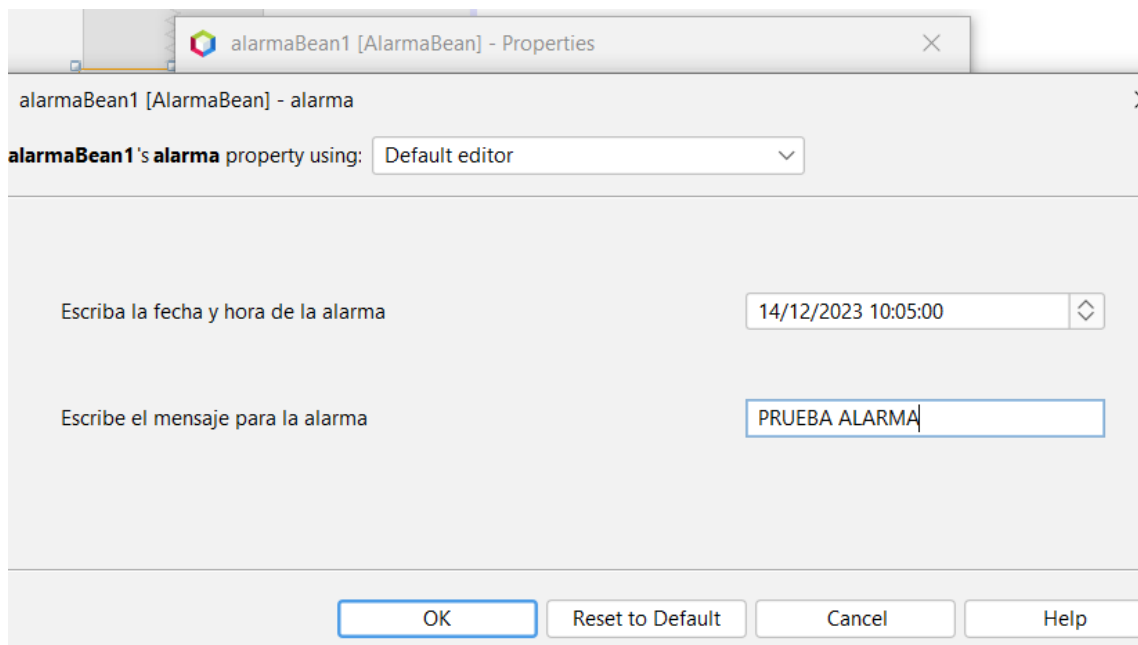
Llegados a este punto, se puede **ejecutar "Clean and Build"** o "Limpiar y Construir" sobre la clase `termometro.java` y agregarlo a la categoría de la paleta de componentes que consideremos oportuna.

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

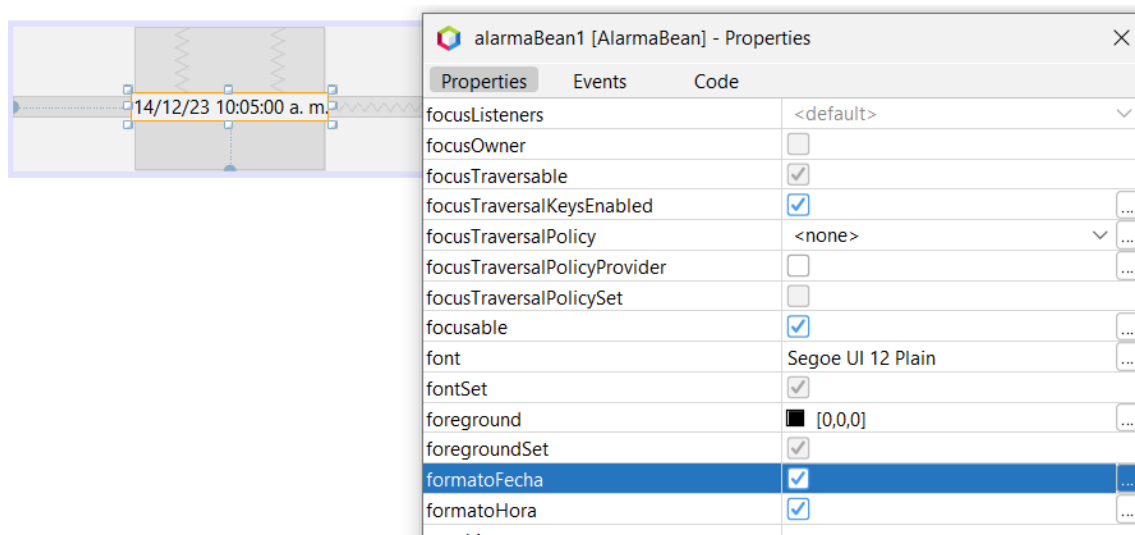
Imagen con el componente insertado tomando hora y fecha sistema, por defecto toma formato fecha larga y formato 12 hora :



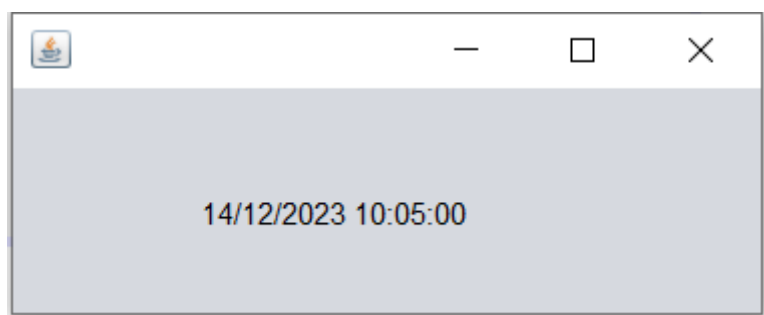
Configuramos nuestra alarma desde las propiedades de nuestro componente, para que salte a las 10:05:00



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.

