2022

Tarea 3

DESARROLLO DE INTERFACES

IRENE CALDELAS FERNÁNDEZ

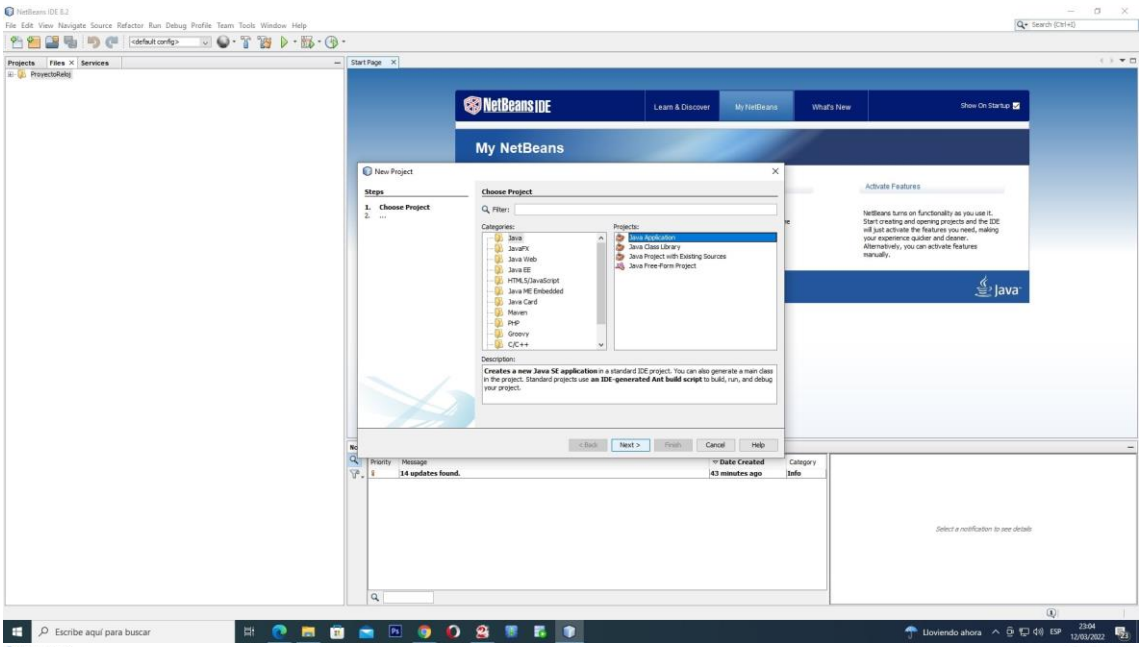
Enunciado.

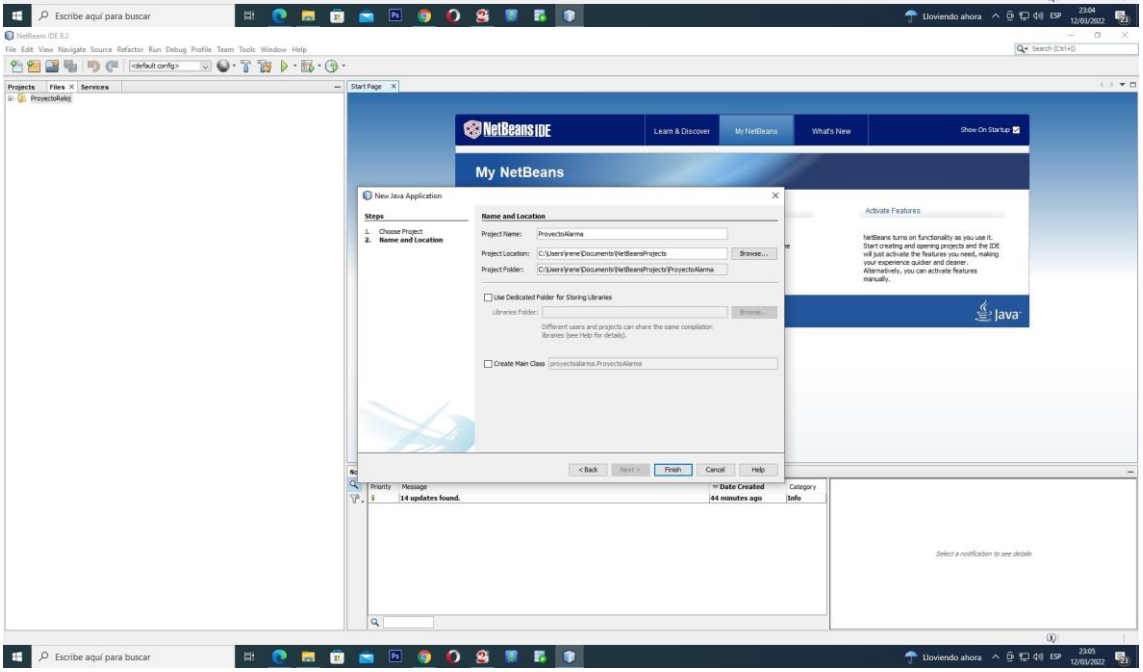
Los chicos de BK están aprendiendo a hacer componente Java usando la plataforma NetBeans. Ahora están intentando crear un reloj digital que poder insertar en cualquier interfaz, el reloj debe tener al menos las siguientes características:

- Una propiedad booleana para indicar si el formato es de 12 o 24 horas.
- Una propiedad booleana para indicar si queremos activar una alarma. El funcionamiento de la alarma consistirá en que se podrá configurar el componente para que a una determinada hora nos muestre un mensaje.
- Dos propiedades para determinar la hora y minuto para el cual queremos programar la alarma. Ambas propiedades será de tipo entero.
- Una propiedad para configurar el mensaje de texto, que queremos que se muestre, cuando se produzca el salto de la alarma. Esta propiedad será de tipo texto (String).
- Función de alarma, si se programa a una hora, debe generar un evento cuando se llegue a esa hora.

Tendrás que crear un formulario de prueba en el que añadas el reloj digital, modifiques el formato de visionado y añadas una alarma para probar que funciona.

Lo primero que vamos a hacer es crear el componente. Para ello vamos a crear un nuevo proyecto de tipo Java Aplicattion a la que vamos a llamar ProyectoAlarma. Deseleccionamos la opción de crear la clase main.





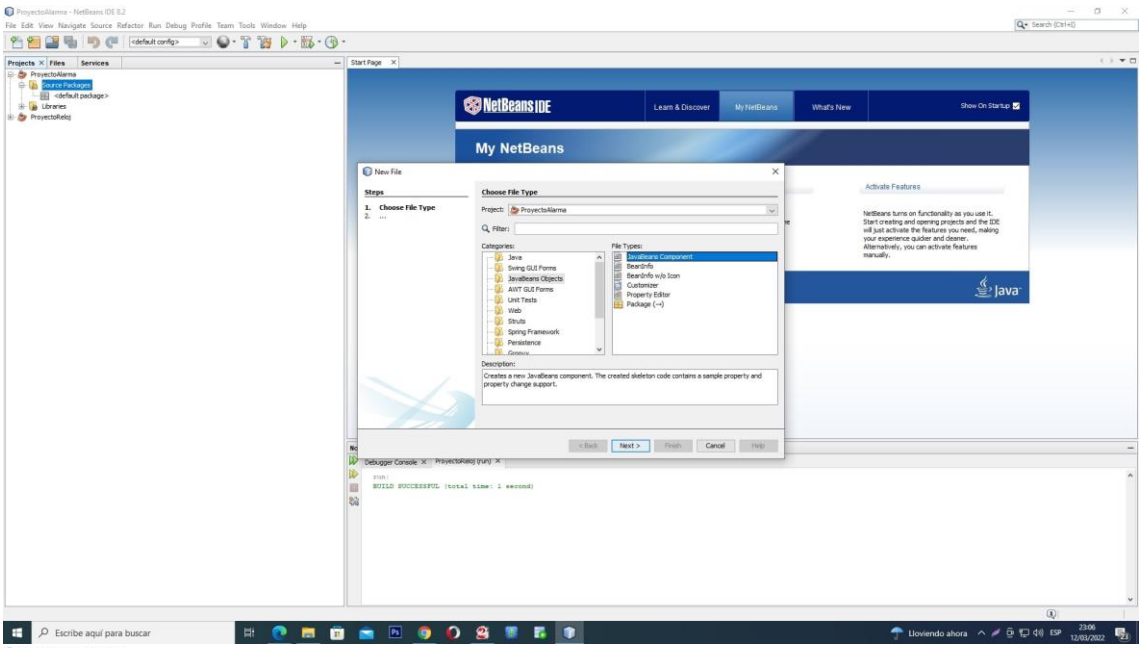
A continuación dentro de del proyecto vamos a crear un componente de JavaBean al que vamos a llamar TemporizadorBean y lo vamos a guardar dentro de un package que vamos a crear con el nombre de temporizador.

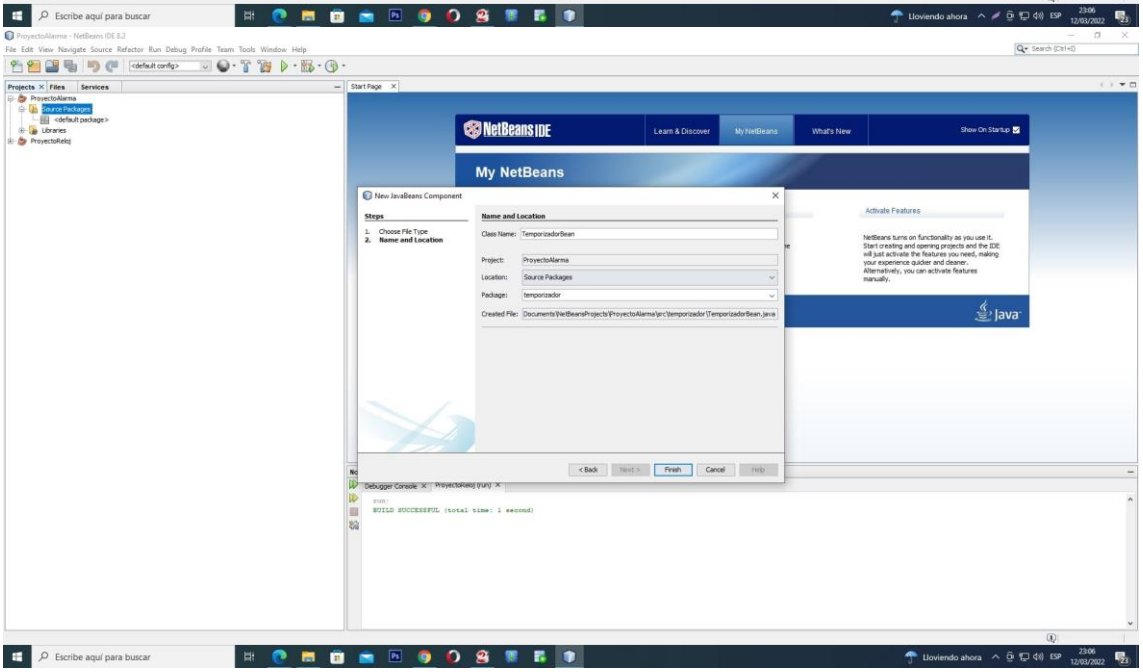
Para que sea considerada un componente tiene que tener implementada la interfaz serializable y además el constructor no debe tener argumentos. Como necesitamos una etiqueta donde se muestra el reloj extenderemos nuestra clase a JLabel. Por último quitaremos todos los métodos que se nos crearon dejando únicamente el constructor.

Posteriormente vamos a proceder a añadir las propiedades que nos piden:

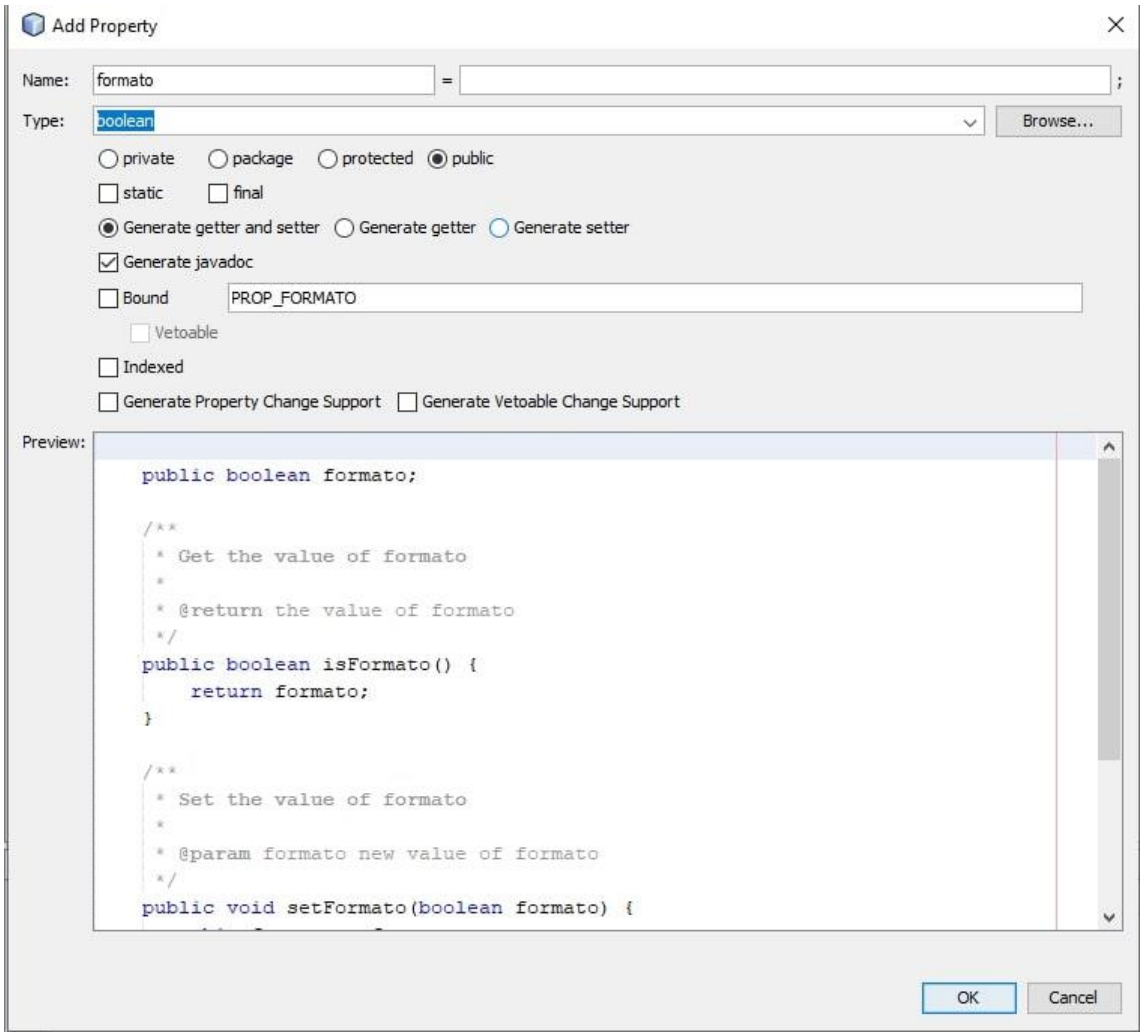
- Formato nos permitirá decidir si queremos un formato de 12 o 24 horas en el reloj.
- activaAlarma será un boolean que nos indicará si la alarma está en fuincionamiento o no.

- horaAlarma guardará el valor de la hora de la alarma.
- minutosAlarma almacenará el valor del minuto en el que queremos que suene la alarma.
- mensajeAlarma guardará el mensaje.
- Tiempo nos va a permitir operar con horas. Al principio lo implementé como un entero pero posteriormente lo cambié a un LocalTime.

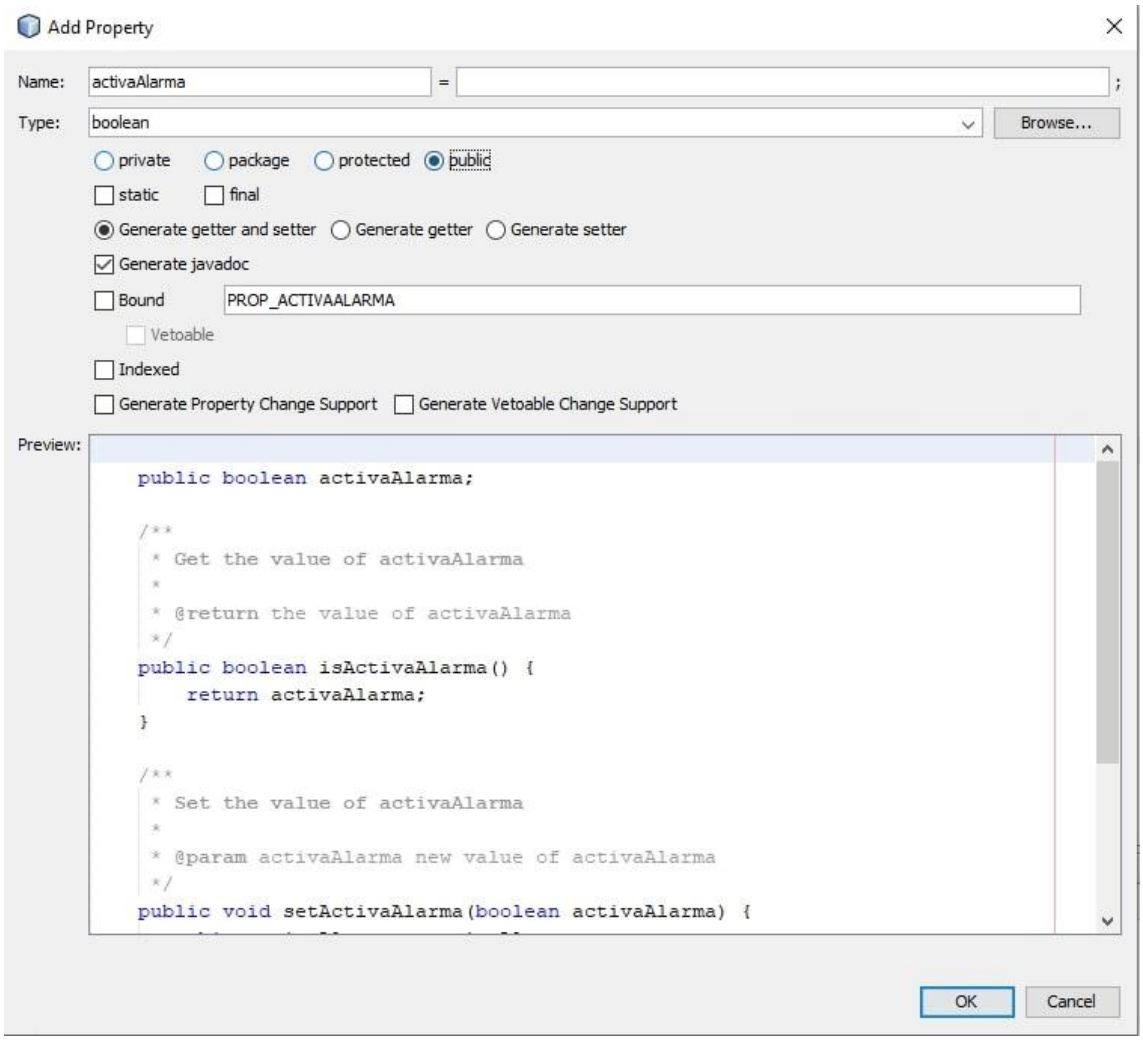




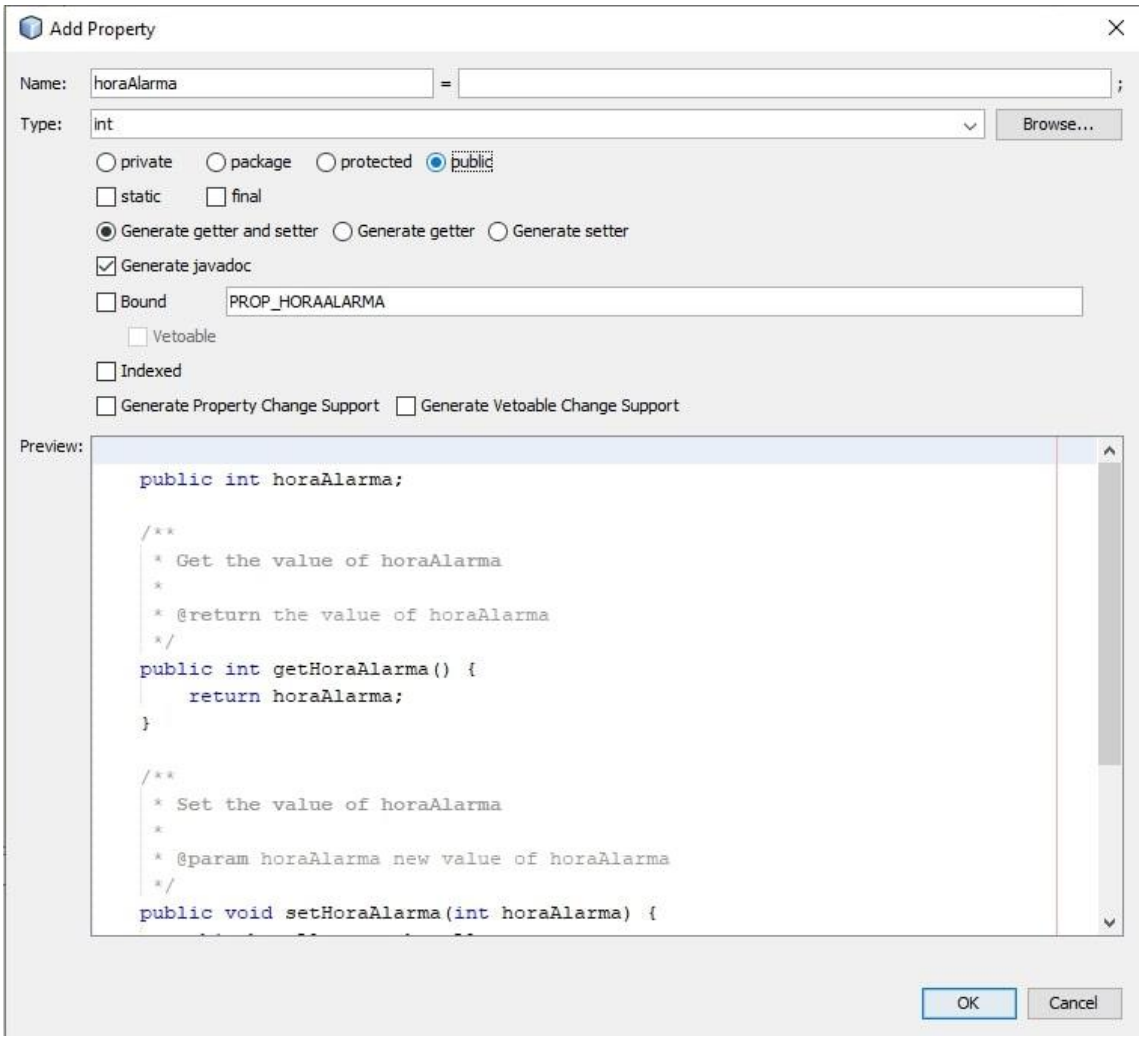
```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.beans.*;
import java.io.Serializable;
import javax.swing.JLabel;

/**
 *
 * @author irene
 */
public class TemporizadorBean extends JLabel implements ActionListener, Serializable {



    public TemporizadorBean() {

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }


}
```



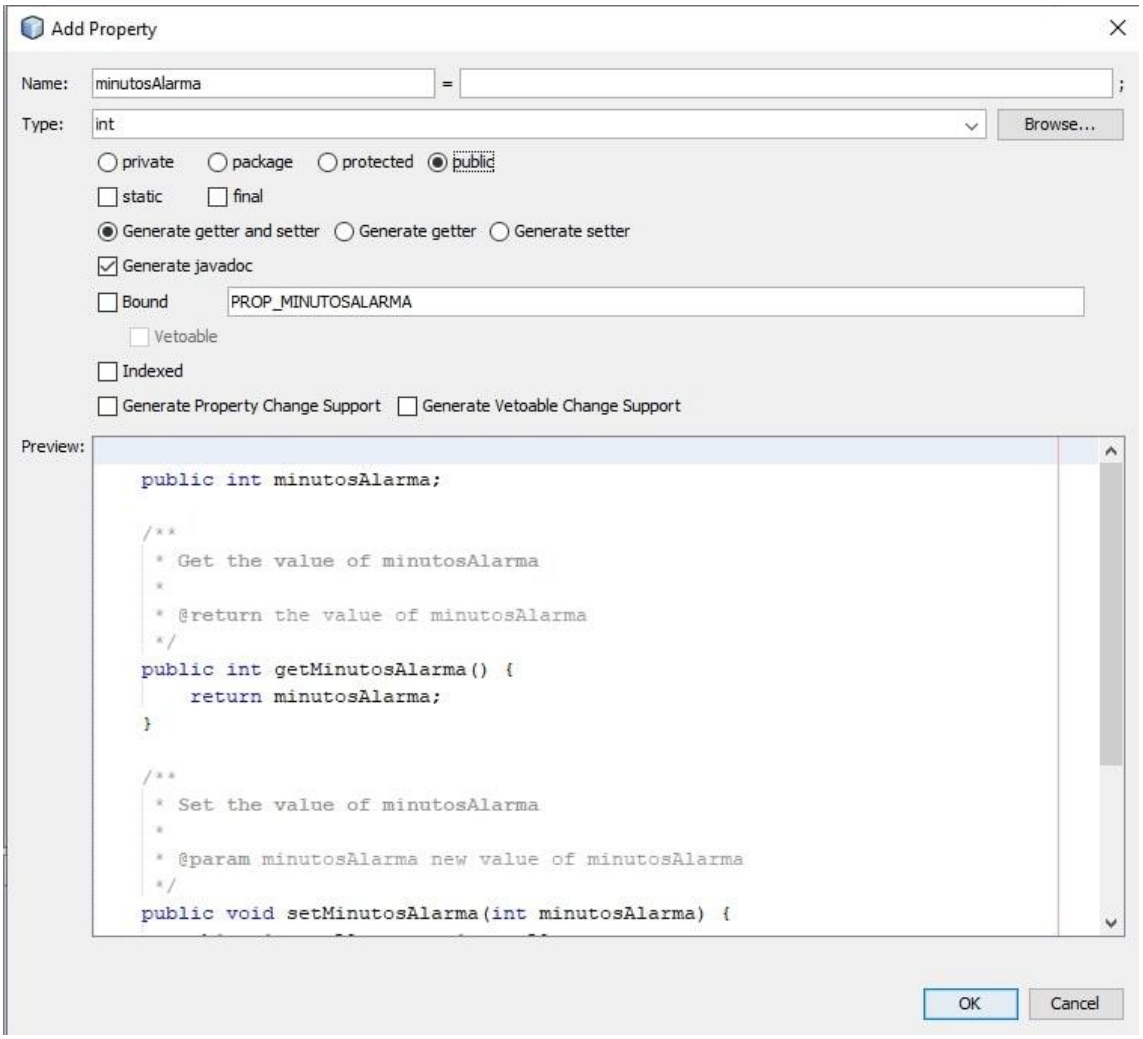
Tarea 3


Add Property
Name: activaAlarma = ;
Type: boolean Browse...
private package protected public
static final
Generate getter and setter Generate getter Generate setter
Generate javadoc
Bound PROP_ACTIVAALARMA
Vetoable
Indexed
Generate Property Change Support Generate Vetoable Change Support
Preview:
public boolean activaAlarma;
/**
* Get the value of activaAlarma
*
* @return the value of activaAlarma
*/
public boolean isActivaAlarma() {
return activaAlarma;
}
/**
* Set the value of activaAlarma
*
* @param activaAlarma new value of activaAlarma
*/
public void setActivaAlarma(boolean activaAlarma) {
OK
Cancel

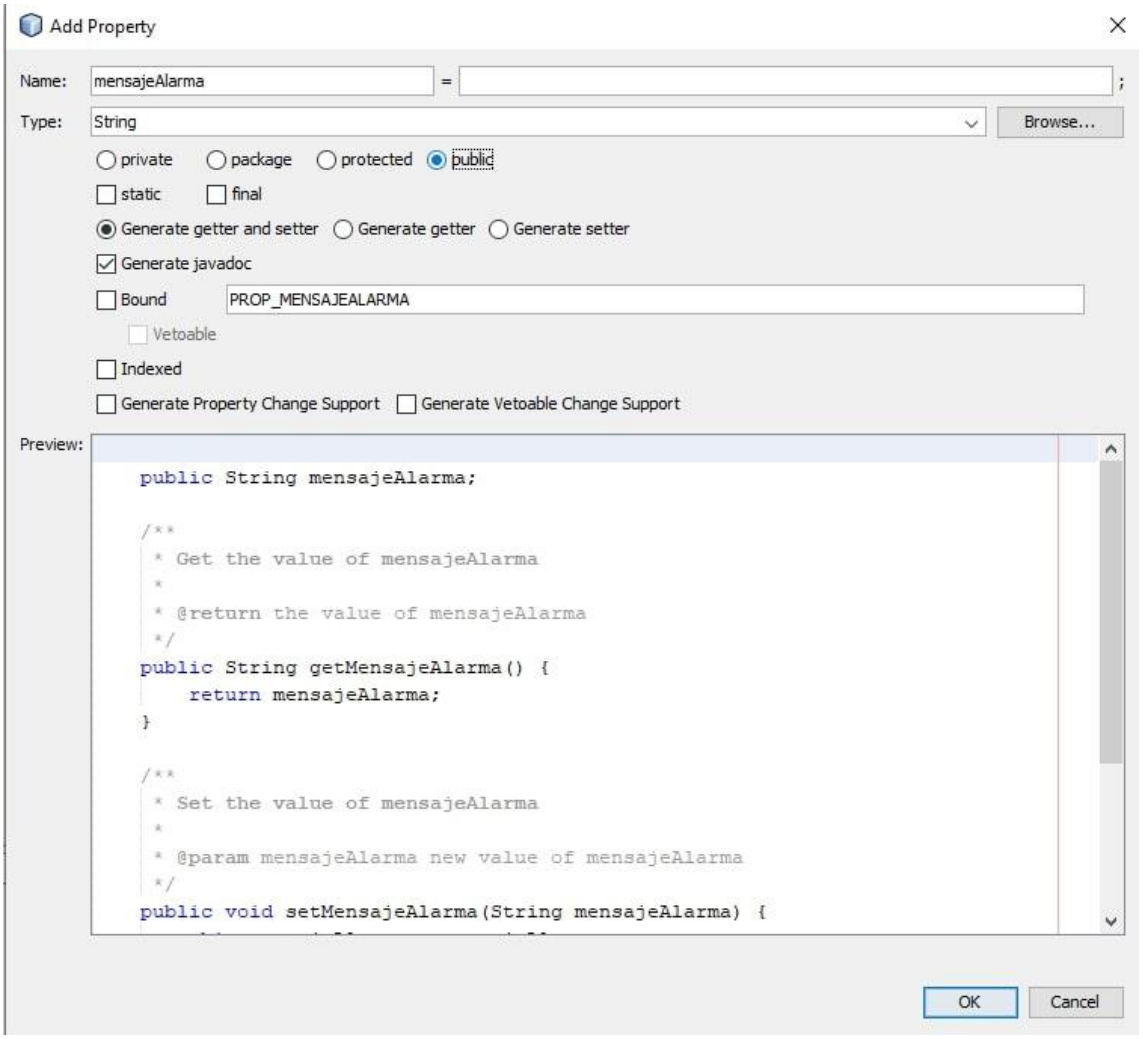
Tarea 3

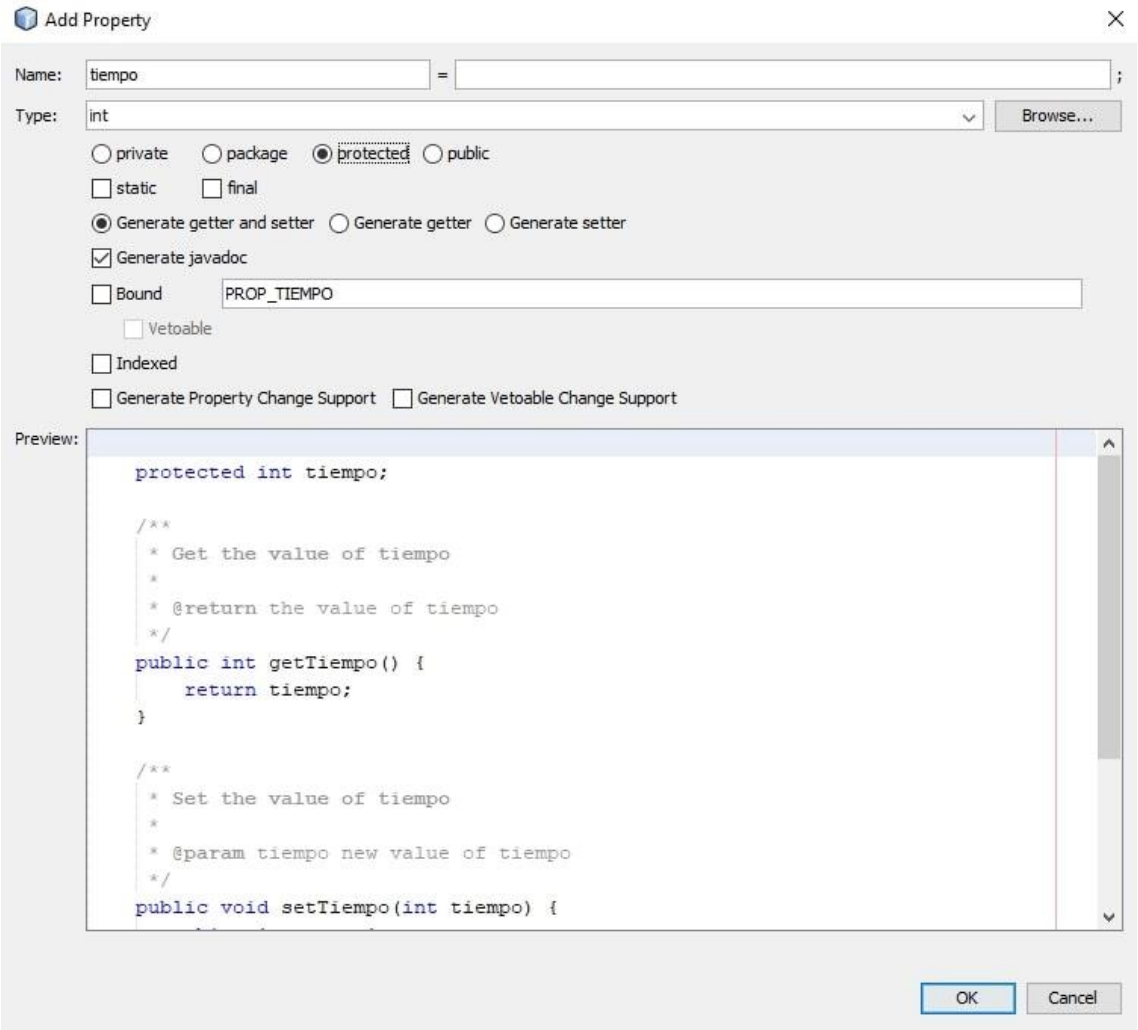

Add Property
Name: horaAlarma = ;
Type: int Browse...
private package protected public
static final
Generate getter and setter Generate getter Generate setter
Generate javadoc
Bound PROP_HORAALARMA
Vetoable
Indexed
Generate Property Change Support Generate Vetoable Change Support
Preview:
public int horaAlarma;
/**
* Get the value of horaAlarma
*
* @return the value of horaAlarma
*/
public int getHoraAlarma() {
return horaAlarma;
}
/**
* Set the value of horaAlarma
*
* @param horaAlarma new value of horaAlarma
*/
public void setHoraAlarma(int horaAlarma) {
OK
Cancel

Tarea 3


Add Property
Name: minutosAlarma = ;
Type: int Browse...
private package protected public
static final
Generate getter and setter Generate getter Generate setter
Generate javadoc
Bound PROP_MINUTOSALARMA
Vetoable
Indexed
Generate Property Change Support Generate Vetoable Change Support
Preview:
public int minutosAlarma;
/**
* Get the value of minutosAlarma
*
* @return the value of minutosAlarma
*/
public int getMinutosAlarma() {
return minutosAlarma;
}
/**
* Set the value of minutosAlarma
*
* @param minutosAlarma new value of minutosAlarma
*/
public void setMinutosAlarma(int minutosAlarma) {
OK
Cancel

Tarea 3


Add Property
Name: mensajeAlarma = ;
Type: String Browse...
private package protected public
static final
Generate getter and setter Generate getter Generate setter
Generate javadoc
Bound PROP_MENSAJEALARMA
Vetoable
Indexed
Generate Property Change Support Generate Vetoable Change Support
Preview:
public String mensajeAlarma;
/**
* Get the value of mensajeAlarma
*
* @return the value of mensajeAlarma
*/
public String getMensajeAlarma() {
return mensajeAlarma;
}
/**
* Set the value of mensajeAlarma
*
* @param mensajeAlarma new value of mensajeAlarma
*/
public void setMensajeAlarma(String mensajeAlarma) {
OK
Cancel




```java
protected LocalTime tiempo;

/**
 * Get the value of tiempo
 *
 * @return the value of tiempo
 */
public LocalTime getTiempo() {
    return tiempo;
}

/**
 * Set the value of tiempo
 *
 * @param tiempo new value of tiempo
 */
public void setTiempo(LocalTime tiempo) {
    this.tiempo = tiempo;
}
```

Nos quedaría algo así al principio, aunque como posteriormente se va a ir viendo fui cambiando algunos tipos.

```
private Timer t;
public boolean formato;
public boolean activaAlarma;
public int horaAlarma;
public int minutosAlarma;
public String mensajeAlarma;
protected LocalTime tiempo = LocalTime.now();
DateTimeFormatter formatoHora;
DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("HH:mm");
```

En el caso del formato puse la condición de que si era true el formato fuera de 24 horas, en caso de false se pondría un formato de 12 horas. Para lograr el formato de 12 horas hice que si la hora era mayor a las 12 del mediodía se le restara 12, aunque debería haber metido este valor en una variable. Posteriormente cree un nuevo formato.

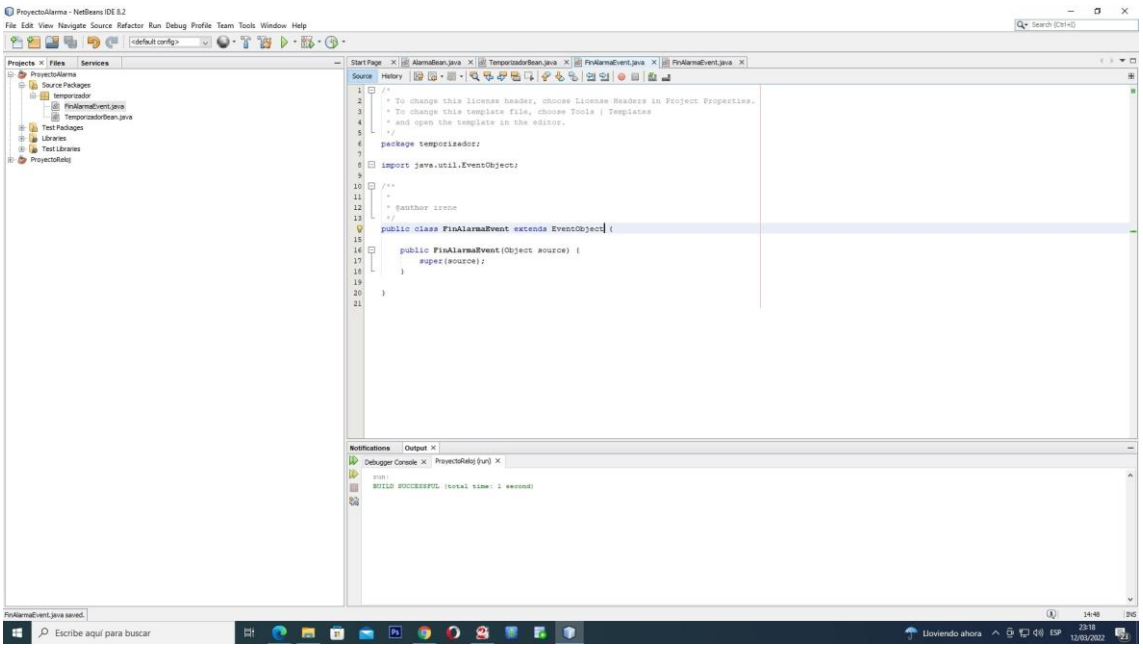
```
public void setFormato(boolean formato) {
    if(formato == true){
       setText(tiempo.format(dateTimeFormatter));
        }else{
        if(tiempo.getHour() > 12){
            int hora = tiempo.getHour() - 12;
            String horaFormateada = Integer.toString(hora);
            formatoHora = DateTimeFormatter.ofPattern(horaFormateada+":"+"mm", Locale.ROOT);
            setText(tiempo.format(formatoHora));
            repaint();
        }
    }
}
```

Vamos a crear un Timer que lo inicializaremos a 1000 para que se ejecute el método actionPerformed cada segundo. Para tener este método tendremos que implementar la interfaz ActionListener.

En setActivaAlarma tendremos el código para gestionar si está funcionando o no. isActivaAlarma nos devolverá si nuestro reloj se está ejecutando o no.

```
public boolean isActivaAlarma() {
    return t.isRunning();
}

/**
 * Set the value of activaAlarma
 *
 * @param activaAlarma new value of activaAlarma
 */
public void setActivaAlarma(boolean activaAlarma) {
     if(activaAlarma == true){
        t.start();
    }else{
        t.stop();
    }
}
```

A continuación vamos a crear una clase que implemente el evento. Esta clase heredará de EventObject y la llamaremos FinAlarmaEvent.



Ahora lo que haremos será definir una interfaz que defina los métodos a usar cuando se genere el evento . Implementará un EventListener. En la imagen se ve como se me olvidó la herencia pero posteriormente la añadí. Esta interfaz tendrá un método al que llamaremos capturaFinAlarma.

Añadiremos también los métodos addFinAlarmaListener y removeFinAlarmaListener que permitirán al componente añadir oyentes y eliminarnos.

En este punto también deberíamos empezar a tocar el método actionPermofed pero preferí ir viendo si el componente iba funcionando antes de implementar el evento.

```
public interface FinAlarmaListener{
    public void capturarFinAlarma(FinAlarmaEvent ev);
}

public void addFinAlarmaListener(FinAlarmaListener receptor){
    this.receptor = receptor;
}

public void removeFinAlarmaListener(FinAlarmaListener receptor){
    this.receptor=null;
}
```

A continuación, vemos un poco como será la interfaz. Tendremos el componente y debajo un spinner de fecha y hora para que seleccionemos a qué hora queremos la alarma y un checkbox que activará la alarma.

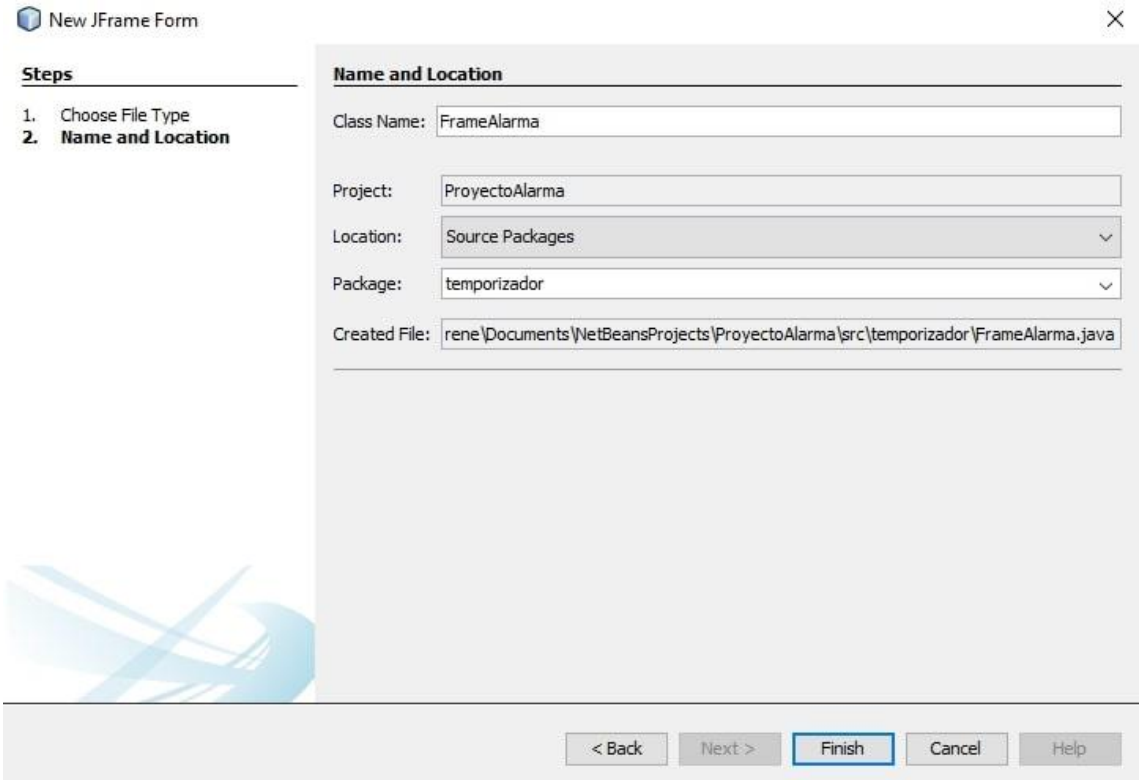
En esta interfaz tendremos un método que nos devolverá la hora del spinner que seleccione el usuario. Nos devolverá un LocalTime.

```
public LocalTime getHoraAlarma() {

    if (spinnerHora.getValue() instanceof Date) {
        Date dateActual = (Date) spinnerHora.getValue();
        return dateActual.toInstant().atZone(ZoneId.systemDefault()).toLocalDateTime().toLocalTime();

    }
    return null;
}
```

Para poder añadir nuestro componente tenemos que limpiar y construir el proyecto y en TemporizadorBean > BeanInfo Editor y lo guardamos.

Tras esto también en TemporizadorBean > Tools > Add to palette para poder arrastrar nuestro componente a la interfaz.

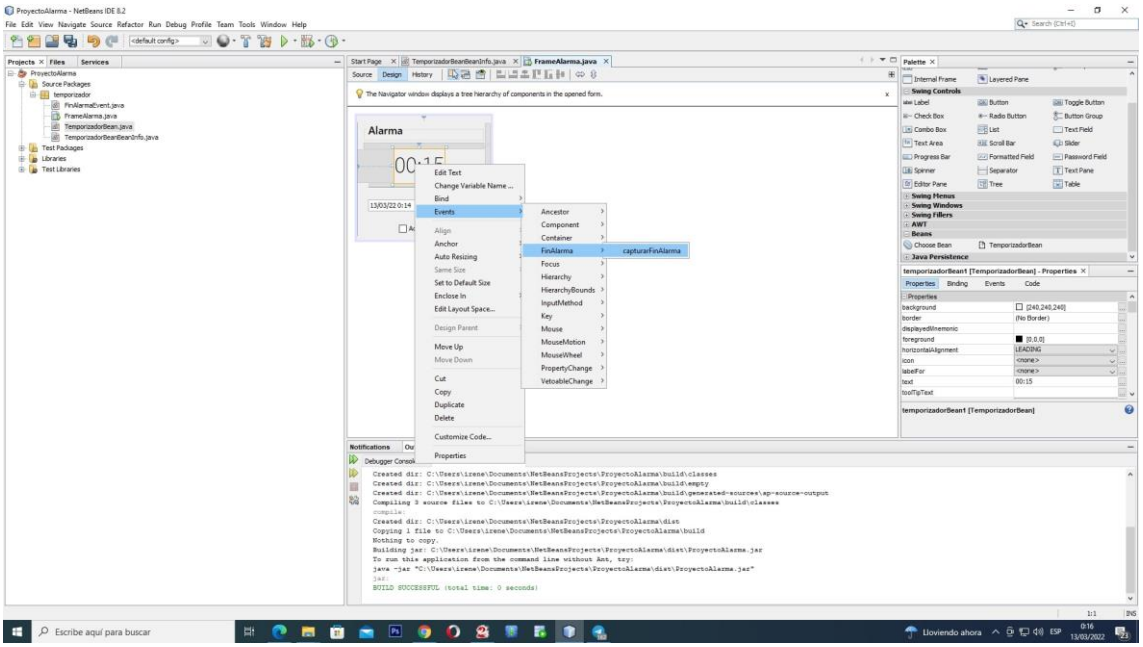
Aquí quise mostrar que la interfaz la hice en un JFrame.



Aquí añado como dije anteriormente la herencia a la interfaz.

```
public interface FinAlarmaListener extends EventListener{
    public void capturarFinAlarma(FinAlarmaEvent ev);
}
```

Para añadir el evento de finalización solo tenemos que seleccionar el componente > Events > FinAlarma > capturaFinAlar



Añadiremos un JOptionPane para que salte cuando se de la alarma.

```
private void temporizadorBean1CapturarFinAlarma(temporizador.FinAlarmaEvent evt) {
    JOptionPane.showMessageDialog(null, "Ring, ring, la alarma está sonando", "Aviso", JOptionPane.INFORMATION_MESSAGE);
}
```

En temporizadorBean tendremos un método que nos permita comparar la hroa actual del sistema y la seleccionada por el usuario. Cuando ambas coincidan devolverá un true que lo recogeremos para que en actionPermofed pueda dar lugar al evento.

```
public boolean alarmaActivada() {

    LocalTime horaActual;

    if (fa == null) {
        return false;
    }

    horaDeAlarma = fa.getHoraAlarma();
    horaAlarma = horaDeAlarma.getHour();
    minutosAlarma = horaDeAlarma.getMinute();
    horaActual = LocalTime.now();

    if (horaActual.getHour() == horaAlarma && horaActual.getMinute() == minutosAlarma) {
        return true;
    }
    return false;

}
```

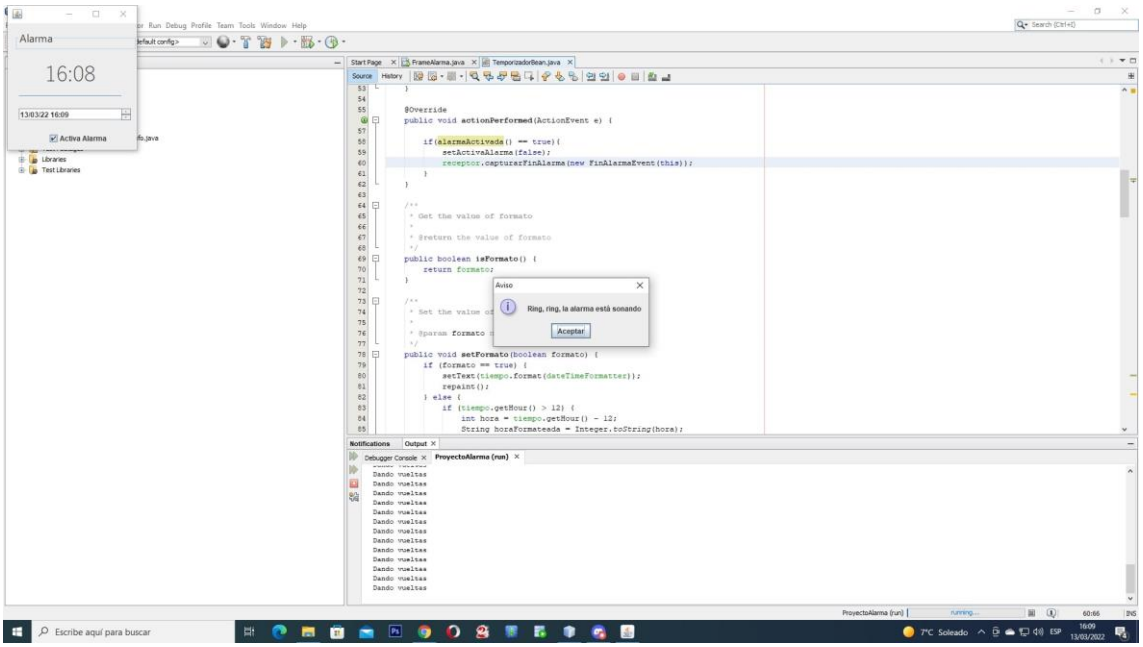
El constructor quedaría tal que así y el actionPerformed pondrá a false el activaAlarma para que el timer pare cuando salte la alarma.

```
public TemporizadorBean() {
    t = new Timer(1000, this);
    setText(tiempo.format(dateTimeFormatter));
}

@Override
public void actionPerformed(ActionEvent e) {

    if (alarmaActivada() == true) {
        setActivaAlarma(false);
        receptor.capturarFinAlarma(new FinAlarmaEvent(this));
    }
}
```

En mi caso he querido que el activaAlarma se ponga a true en el momento en el que se selecciona el checkbox de la interfaz. En caso de quitarlo se pararía. También a la hora de seleccionar la hora salte un mensaje diciendo que se ha configurado una alarma.

```
private void checkAlarmaActionPerformed(java.awt.event.ActionEvent evt) {
    if (checkAlarma.isSelected()) {
        temporizadorBean1.setActivaAlarma(true);
        JOptionPane.showMessageDialog(null, "Alarma configurada", "Aviso", JOptionPane.INFORMATION_MESSAGE);
    } else {
        temporizadorBean1.setActivaAlarma(false);
    }
}
```

Para ver que se hacía bien el salto de la alarma puse un system.out.println en el método alarmaActivada para ver si el tiempo avanzaba y se paraba en el momento en el que saltara la alarma como se ve a continuación. Posteriormente limpié el código y quité estos println.



En el caso del formato como mencioné anteriormente en caso de que fuera true el formato sería de 24 horas. En cambio si formato estaba a false el formato del reloj sería de 12 horas.