Tarea 3

Informe del Proyecto: Reloj Digital con Alarma

1. Introducción

Este proyecto tiene como objetivo desarrollar un componente en Java para representar un reloj digital con la capacidad de configurar y gestionar alarmas. Este componente se implementa como un JPanel personalizable, que puede ser fácilmente integrado en otras interfaces gráficas en la plataforma NetBeans. El reloj debe permitir al usuario seleccionar el formato de hora (12 o 24 horas), configurar una alarma a una hora específica y mostrar un mensaje cuando la alarma se active.

El proyecto está enfocado en el manejo de eventos y la creación de componentes gráficos reutilizables, utilizando propiedades que permiten modificar el comportamiento del reloj y de la alarma.

2. Descripción del Componente

El componente **RelojDigital** permite lo siguiente:

1. **Formato de hora (12/24 horas):** El usuario puede elegir entre formato de 12 horas o de 24 horas.
2. **Alarma:** El usuario puede configurar la hora y el minuto exacto para la alarma. Además, se puede activar o desactivar la alarma.
3. **Mensaje de la alarma:** El usuario puede definir un mensaje de texto que se mostrará cuando la alarma se active.
4. **Eventos de la alarma:** Cuando la hora y minuto de la alarma coinciden con la hora actual, se genera un evento que notifica al usuario.

3. Elaboración de las clases

I. En primer lugar, se ha creado la clase relojDigital, que será la clase principal encargada de mostrar la hora y gestionar la Alarma. Esta clase hereda de JPanel para que pueda ser insertada en cualquier interfaz gráfica. Además contiene una interfaz Serializable para que pueda ser considerado como un componente.

```
public class RelojDigital extends JPanel implements ActionListener, Serializable {
```

El primer paso fue crear un temporizador que actualiza la hora cada segundo. Para ello se creó un Timer que actualiza el temporizador cada segundo.

Tarea 3

```
// Temporizador que se usará para actualizar el reloj cada segundo
private final Timer temporizador;

// Lista de oyentes de la alarma
private final List<AlarmaListener> alarmaListeners = new ArrayList<>();


//Constructor que inicializa el temporizador para actualizar la hora
// cada segundo y inicializa la alarma como inactiva.
public RelojDigital() {
    temporizador = new Timer(1000, this);// Temporizador que llama a actionPerformed cada segundo
    temporizador.start();

    // Inicializamos la alarma como inactiva
    alarma = new Alarma(null, false);
}
```

Esta clase tiene las siguientes variables:

formato: Propiedad booleana que indica si el reloj debe mostrar la hora en formato de 12 o 24 horas.

tiempo: Objeto LocalTime que almacena la hora actual.

alarma: Objeto Alarma que gestiona la hora y el estado de la alarma.

horaAlarma y minutoAlarma: Enteros que almacenan la hora y el minuto para configurar la alarma.

mensajeAlarma: Cadena de texto (String) que se muestra cuando la alarma se activa.

Así mismo se han incluido un DateTimeFormater para establecer la hora en un formato de 12 o 24 horas.

```
public class RelojDigital extends JPanel implements ActionListener, Serializable {

    private boolean formato12_24;  // Para indicar el formato de hora (12 o 24 horas)
    private LocalTime tiempo; // Guarda la hora actual
    private Alarma alarma;    // Objeto de tipo Alarma que contiene la hora y estado de la alarma
    private int horaAlarma;   // Hora de la alarma (en formato de 24 horas)
    private int minutoAlarma; // Minuto de la alarma
    private String mensajeAlarma; // Mensaje que se muestra cuando la alarma se activa

    // Formatos de hora para 12 y 24 horas
    private final DateTimeFormatter formato12Horas = DateTimeFormatter.ofPattern("hh:mm:ss a");
    private final DateTimeFormatter formato24Horas = DateTimeFormatter.ofPattern("HH:mm:ss");

    // Temporizador que se usará para actualizar el reloj cada segundo
    private final Timer temporizador;

    // Lista de oyentes de la alarma
    private final List<AlarmaListener> alarmaListeners = new ArrayList<>();
```

Así mismo se han introducido los métodos getter y setter que permiten acceder y modificar las variables:

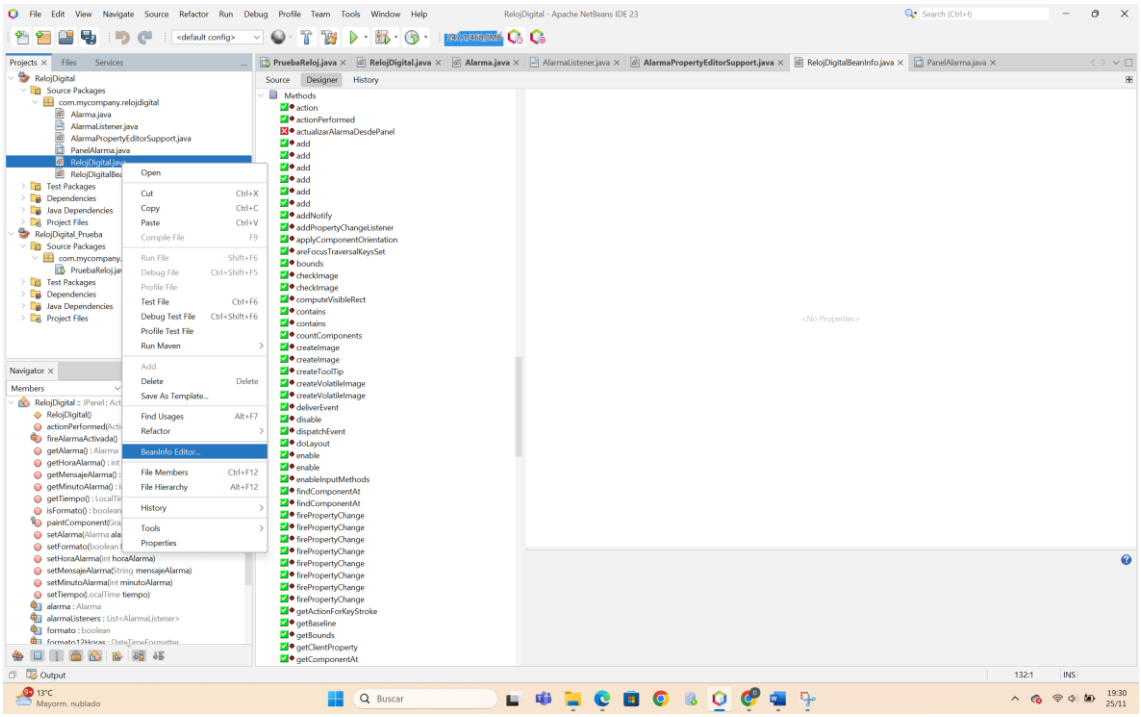
Tarea 3

```
public Alarma getAlarma() {
    return alarma;
}

public void setAlarma(Alarma alarma) {
    this.alarma = alarma;

    if (alarma != null && alarma.getHoraAlarma() != null) {
        this.horaAlarma = alarma.getHoraAlarma().getHours();
        this.minutoAlarma = alarma.getHoraAlarma().getMinutes();
    }
}

public boolean isFormato() {
    return formato;
}

public void setFormato(boolean formato) {
    this.formato = formato;
    repaint();
}

public int getHoraAlarma() {
    return horaAlarma;
}

public void setHoraAlarma(int horaAlarma) {
    this.horaAlarma = horaAlarma;
}

public int getMinutoAlarma() {
    return minutoAlarma;
}

public void setMinutoAlarma(int minutoAlarma) {
    this.minutoAlarma = minutoAlarma;
}

public String getMensajeAlarma() {
    return mensajeAlarma;
}

public void setMensajeAlarma(String mensajeAlarma) {
    this.mensajeAlarma = mensajeAlarma;
}
```

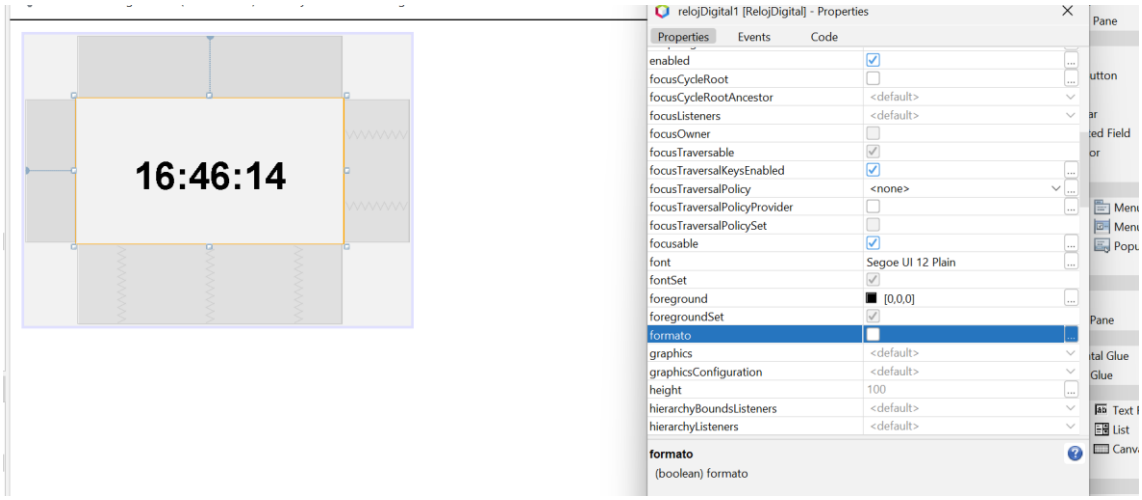
Tarea 3

Para mostrar la hora en el panel, se utilizó el método paintComponent que sobrescribe el método de la clase base JPanel. Este método se encarga de dibujar la hora en el panel en formato 12 o 24 horas:

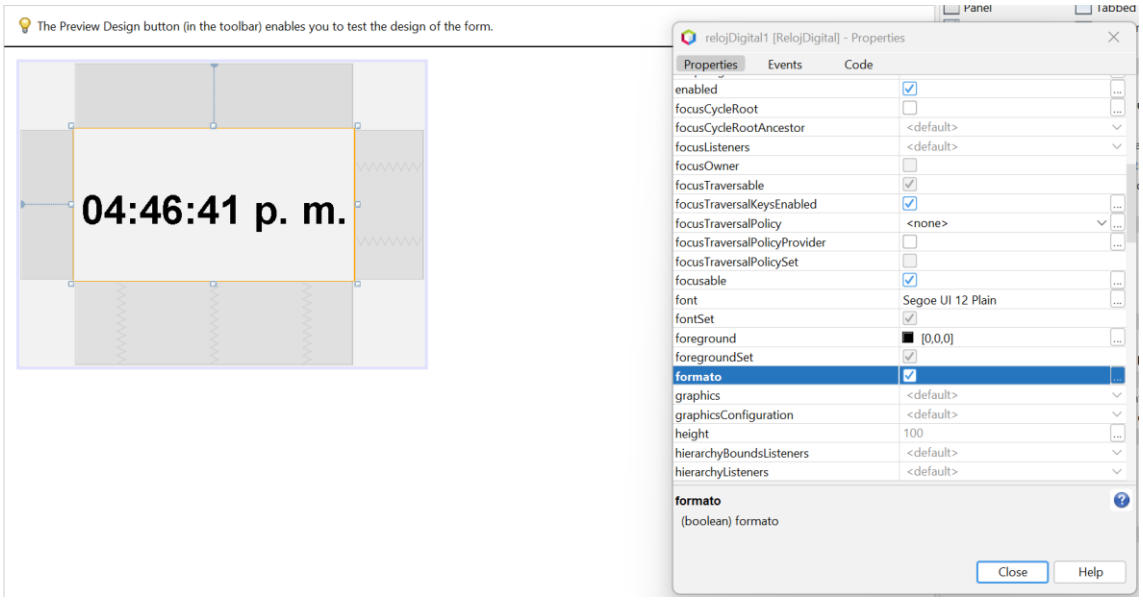
Tras esto procedemos a crear el BeanInfo de la clase reloj y comprobamos el funcionamiento del reloj, así como de la propiedad formato.



Para ello se ha creado un nuevo proyecto llamado RelojDigital_Prueba en el que se ha incluido un jFrame en el que se ha incluido este componente:



Tarea 3



II. Una vez creado el reloj y comprobar que funciona correctamente su función cambiar formato, se procede a configurar la alarma. Para ello se ha creado la clase Alarma.

Esta clase tiene las siguientes variables:

- horaAlarma: variable de tipo Date para almacenar la hora y minuto de la alarma.
- activa: variable de tipo boolean que indica si la alarma está activa (true) o no (false).

El constructor de esta clase inicializa los atributos de la alarma con la hora de activación y el estado de activación.

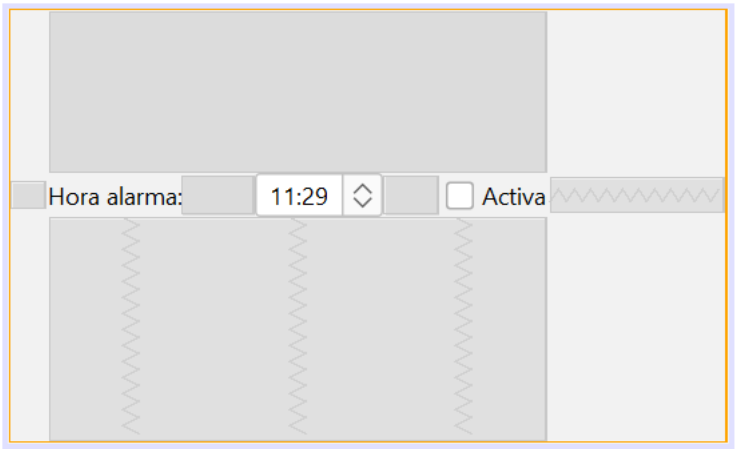
```
package com.mycompany.relojdigital;

import java.io.Serializable;
import java.util.Date;
import javax.swing.JLabel;

/**
 * La clase Alarma representa una alarma que contiene la hora de activación y un
 * estado que indica si la alarma está activada o desactivada. Esta clase
 * extiende de JLabel para poder ser utilizada en interfaces gráficas.
 */
public class Alarma extends JLabel implements Serializable {

    // Hora a la que debe sonar la alarma
    private Date horaAlarma;
    // Estado que indica si la alarma está activa o no
    private boolean activa = false;

    /**
     * Constructor que inicializa una nueva alarma con una hora y un estado de
     * activación.
     */
    public Alarma(Date horaAlarma, boolean activa) {
        this.horaAlarma = horaAlarma;
        this.activa = activa;

    }
```

Tarea 3

Además implementa los métodos get y set de ambos parámetros.

```
public Date getHoraAlarma() {
    return horaAlarma;
}

/**
 * Establece la hora de la alarma.
 * @param horaAlarma La nueva hora a la que debe sonar la alarma.
 */
public void setHoraAlarma(Date horaAlarma) {
    this.horaAlarma = horaAlarma;
}

/**
 * Obtiene el estado de la alarma, si está activa o desactivada. devuelve
 * true si la alarma está activa y false si está desactivada.
 */
public boolean isActiva() {
    return activa;
}

/**
 * Establece el estado de la alarma.
 * @param activa El nuevo estado de la alarma: true para activarla, false
 * para desactivarla.
 */
public void setActiva(boolean activa) {
    this.activa = activa;
}
```

III. Para introducir la propiedad alarma en el componente RelojDigital, primeramente se va a crear un JPanel que incluye un JSpinner en formato fecha y un checkBox que permita seleccionar si la alarma está activa o no. De esta manera se pretende que al dar a la propiedad alarma, se abra este panel para establecer la hora y activar la alarma.



Tarea 3

El constructor de esta clase inicia el componente panel y recoge el valor de la hora y de la activación o no de la alarma:

```
package com.mycompany.relojdigital;

import java.util.Date;

/**
 *
 * @author David
 */
public class PanelAlarma extends javax.swing.JPanel {

    /**
     * Creates new form PanelAlarma
     */
    public PanelAlarma() {

        initComponents();
        SpinnerAlarma.setValue(new Date());
        chkBoxActiva.setSelected(false);
    }
```

Así mismo se ha incluido el método obtenerDatosAlarma() que recoger los valores introducidos en el Spinner y el checkBox y crea una nueva instancia de Alarma proporcionándole estos valores:

```
public Alarma obtenerDatosAlarma(){

    Date date = (Date) SpinnerAlarma.getValue();
    boolean activa = chkBoxActiva.isSelected();
    return new Alarma(date, activa);
}
```

Para poder crear la propiedad se ha credo la clase AlarmaPropertyEditorSupport que extiende de PropertyEditorSupport. Su función es permitir la edición personalizada de la propiedad de tipo Alarma, mostrando un editor gráfico (el PanelAlarma) para manipular las propiedades de la alarma, como la hora y su estado de activación.

Esta clase inicializa un componente panelAlarma e inserta los siguientes métodos:

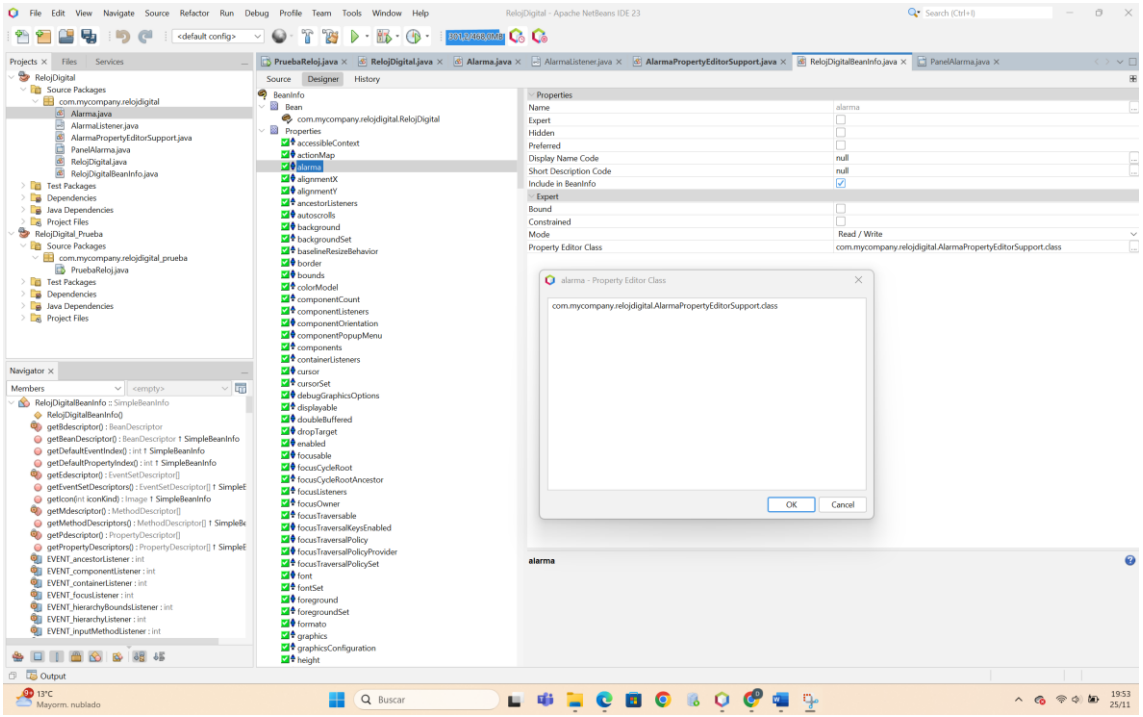
- Método supportsCustomEditor(): Este método indica si el editor personalizado es compatible para editar la propiedad. En este caso, siempre devuelve true porque queremos usar el PanelAlarma como editor.
- Método getCustomEditor(): Este método devuelve el editor personalizado que se utilizará para editar la propiedad. En este caso, devuelve la instancia del PanelAlarma, que es el editor donde el usuario podrá configurar la alarma.
- Método getJavaInitializationString(): Este método devuelve una cadena de texto que representa la inicialización de la propiedad en formato Java. Básicamente, genera un string que puede ser utilizado para crear una nueva instancia de la alarma con los valores configurados en el PanelAlarma. Utiliza el valor de la hora (como un long) y el estado de activación (como un boolean) para generar una cadena que representa la inicialización del objeto Alarma.

Tarea 3

- Método getValue(): Este método devuelve el valor actual de la propiedad, que en este caso es el objeto Alarma que contiene la hora configurada y su estado de activación. Utiliza el método obtenerDatosAlarma() para obtener una nueva instancia de Alarma con los valores actuales del PanelAlarma.

```java
package com.mycompany.relojdigital;

import java.beans.PropertyEditorSupport;
import java.util.Date;


public class AlarmaPropertyEditorSupport extends PropertyEditorSupport {

    private PanelAlarma panel = new PanelAlarma();

    @Override
    public boolean supportsCustomEditor() {
        return true;
    }

    @Override
    public java.awt.Component getCustomEditor() {
        return panel;
    }

    @Override
    public String getJavaInitializationString() {
        Date horaAlarma = panel.obtenerDatosAlarma().getHoraAlarma();
        boolean activa = panel.obtenerDatosAlarma().isActiva();
        return "new com.mycompany.relojdigital.Alarma(new java.util.Date("+ horaAlarma.getTime() +"l),"+ activa+")";
    }

    @Override
    public Object getValue() {
        return panel.obtenerDatosAlarma();
    }
}
```

Para añadir el componente PanelAlarma a la propiedad alarma del componente RelojDigital, accedemos a dicha propiedad e insertamos la clase AlarmaPropertyEditorSupport en el PropertyEditorClass de alarma:



Tarea 3

IV. Para manejar los eventos que se producen a la hora de activarse la alarma se ha implementado la interfaz AlarmaListener. La interfaz actúa como un canal de notificación para los eventos de la alarma. Las clases que implementen AlarmaListener serán notificadas automáticamente cuando la alarma se active, sin necesidad de depender directamente de la clase RelojDigital.
La interfaz tiene un único método:

```
package com.mycompany.relojdigital;

import java.util.EventListener;


public interface AlarmaListener extends EventListener {

    void onAlarmaActivada(String mensaje);

}
```

La clase RelojDigital mantiene una lista de objetos que implementan la interfaz AlarmaListener. Esto permite que cualquier clase que implemente esta interfaz pueda ser registrada como "oyente" de los eventos de alarma. Esta lista se declara así en la clase RelojDigital:

```
// Lista de oyentes que recibirán notificaciones cuando la alarma se active
private final List<AlarmaListener> alarmaListeners = new ArrayList<>();
```

Para notificar a todos los oyentes registrados que la alarma se ha activado se ha creado el método fireAlarmaActivada. Este método recorre la lista alarmaListeners ejecutando el método onAlarmaActivada(String mensaje) de cada objeto registrado, y muestra un cuadro de diálogo emergente (JOptionPane) con el mensaje configurado en la alarma.

Tras esto establece el estado de la alarma en inactiva y llama a repaint() para reflejar cualquier cambio visual en el componente.

```
/**
 * Método que se ejecuta cuando la alarma se activa. Notifica a todos los oyentes
 * y muestra un mensaje emergente con el mensaje de la alarma.
 */
private void fireAlarmaActivada() {
    // Notifica a todos los oyentes de la alarma
    for (AlarmaListener listener : alarmaListeners) {
        listener.onAlarmaActivada(mensajeAlarma);
    }
    // Muestra un mensaje emergente con el mensaje de la alarma
    javax.swing.SwingUtilities.invokeLater(() -> {
        JOptionPane.showMessageDialog(this, mensajeAlarma, "Alarma Activada", JOptionPane.WARNING_MESSAGE);
    });
    alarma.setActiva(false); // Desactiva la alarma tras activarse
     // Repinta el panel para reflejar el cambio
    repaint();
}
```

Por último, se implementa el método actionPerformed. La función de este es la de actualizar la hora actual del reloj, repintar el panel para mostrar la hora actualizada y verificar si la alarma debe activarse. En caso de que se cumpla la condición de que la alarma no sea nula y se encuentre activa se comprueba que la hora actual se encuentre entre el primer segundo y el

último de la hora y minuto establecido y se invoca el método fireAlarmaActivada(), disparando la alarma.

```
@Override
public void actionPerformed(ActionEvent e) {
    // Actualiza la hora cada segundo
    tiempo = LocalTime.now();

    repaint();// Repinta el panel con la nueva hora

    // Si la alarma está activada, verifica si la hora actual coincide con la hora de la alarma
    if (alarma != null && alarma.isActiva()) {
        LocalTime alarmaTime = LocalTime.of(horaAlarma, minutoAlarma);
        //verifica si la hora actual está dentro del intervalo de 1 segundo antes y 59 segundos después de la hora de la alarma.
        //Si la condición se cumple, se activa la alarma.
        if (tiempo.isAfter(alarmaTime.minusSeconds(1)) && tiempo.isBefore(alarmaTime.plusSeconds(59))) {
            fireAlarmaActivada();
        }
    }
}
```

El método ActionPerformed() se invoca automáticamente cada segundo por el temporizador que se inicializa en el constructor de relojDigital como se ha visto anteriormente. A continuación se explica el funcionamiento del constructor:

Se inicializa el Timer temporizador, configurado para generar eventos de acción cada 1000 milisegundos (1 segundo), el segundo argumento, this, indica que la instancia actual de la clase RelojDigital será el receptor de los eventos generados por el temporizador, ya que la clase implementa la interfaz ActionListener.

Tras esto, se inicia el temporizador, haciendo que dispare eventos periódicamente cada segundo.

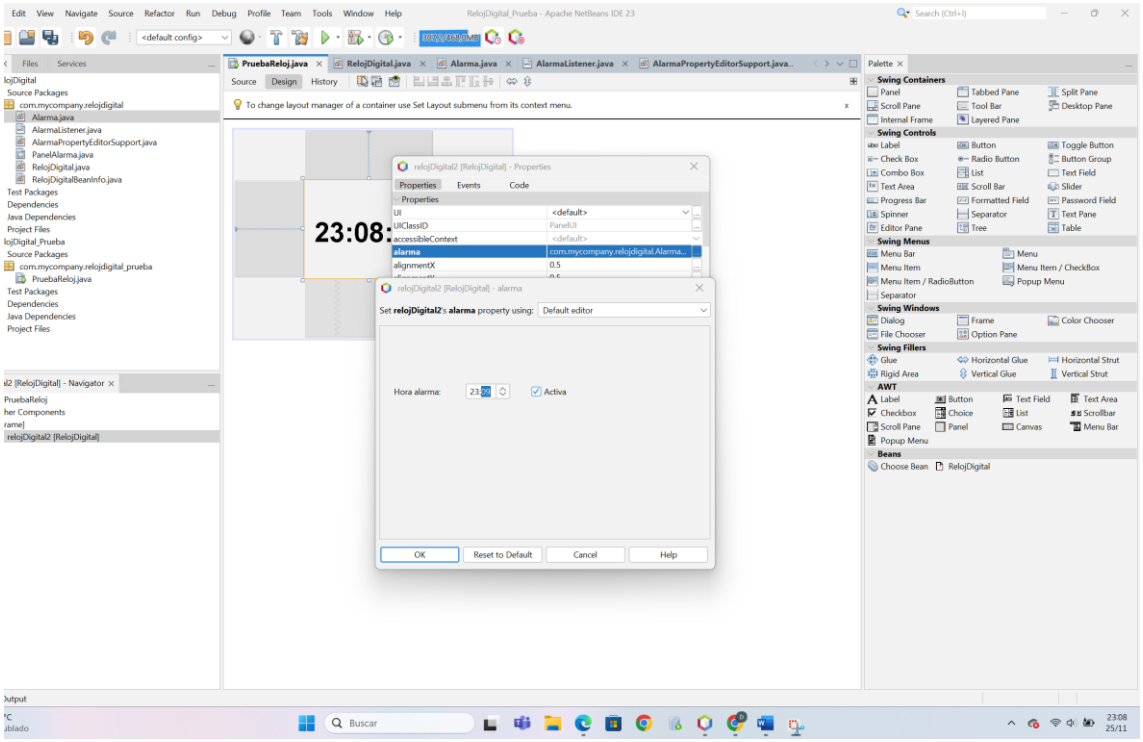
Estos eventos son capturados por el método actionPerformed.

```
/**
 * Constructor que inicializa el temporizador para actualizar la hora
 * cada segundo y configura la alarma como inactiva.
 */
public RelojDigital() {
    temporizador = new Timer(1000, this);
    temporizador.start();

    // Inicializamos la alarma como inactiva
    alarma = new Alarma(null, false);
}
```

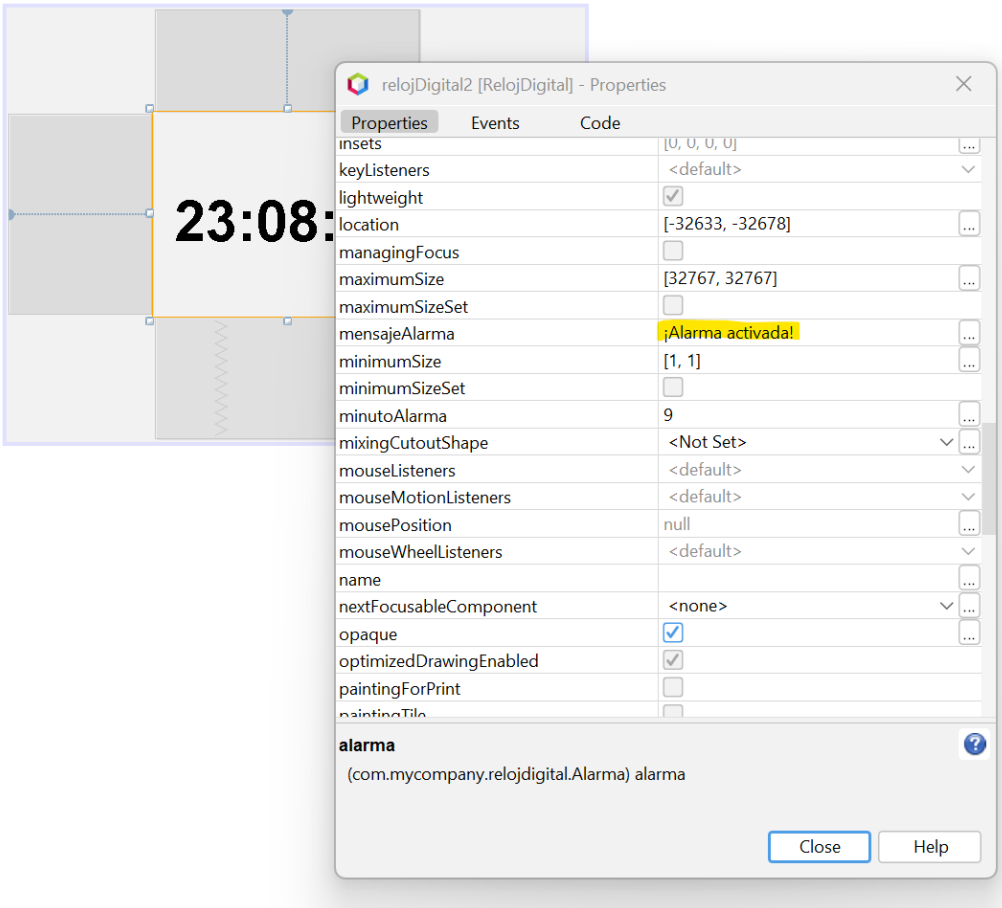
4. Pruebas del funcionamiento del componente.

Como hemos visto anteriormente, para la realización de las pruebas se ha creado un nuevo proyecto RelojDigital_Prueba. En este se ha creado un nuevo JFrame en el que se ha implementado el componente. En propiedades establecemos la alarma:

Tarea 3



También podemos configurar el mensaje que queremos que se muestre al activarse esta alarma. Dejamos el mensaje que estaba ya predefinido:



Tarea 3

Al llegar la hora indicada se dispara el evento mostrando el mensaje de la alarma: