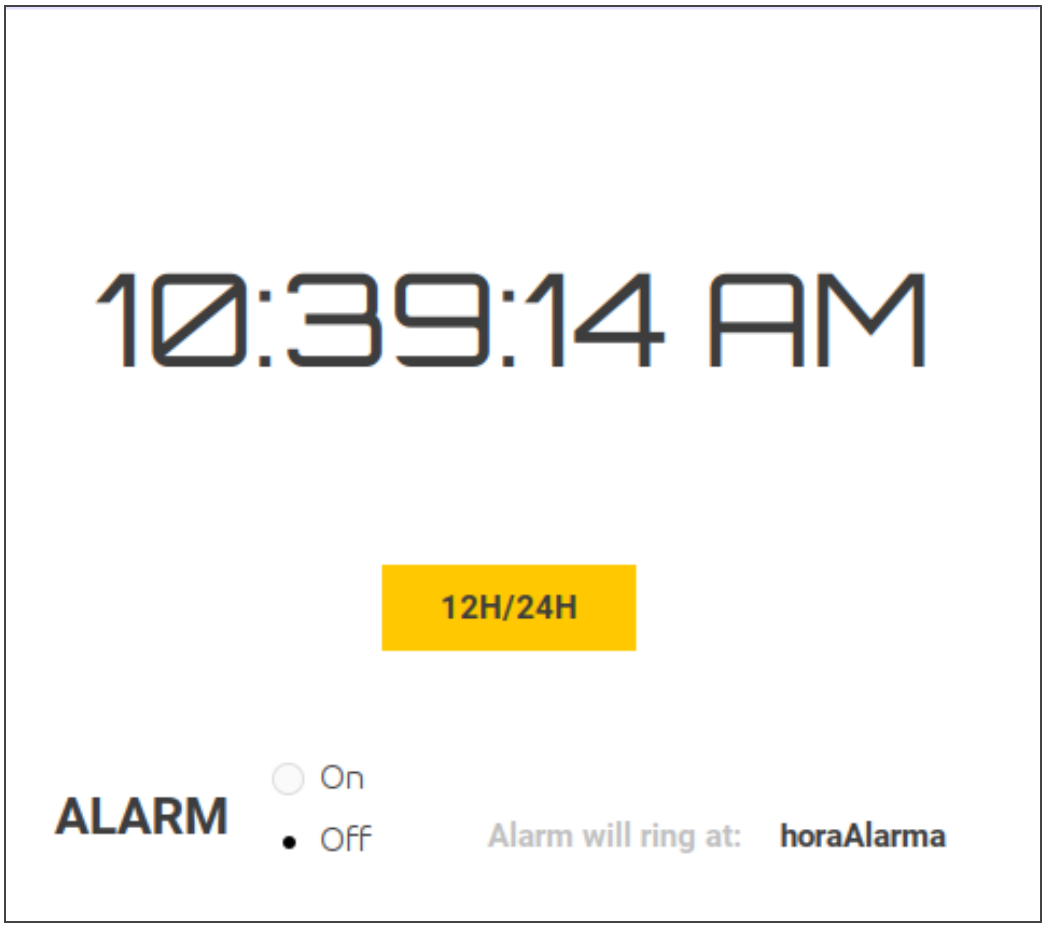
DI03
Componente reloj digital



Francisco Iván Ramírez Ortega

DAD

Objetivo

Generar un componente Java en el IDE Netbeans para su uso en otras interfaces.

Requisitos

- 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.

Criterios de puntuación

- Creación del componente. 1 punto.
- Creación de las diferentes propiedades y sus métodos getters y setters. 2 puntos
- Crear la clase que hereda de EventObject para que se puedan crear los eventos a lanzar. 1 punto.
- Añadir el código necesario para modificar la hora cada segundo que pasa. 2 puntos.
- Añadir el código necesario para generar los eventos cuando se llegue a la hora de la alarma. 2 puntos.
- Generar el ejemplo de prueba del componente en el que se añada, se cambie el formato de visionado y se añadan la alarma. 1 punto.
- Capturar el evento y mostrar un mensaje cuando se produzca una alarma. 1 punto.

Generar un Bean - ClockLabel

El Bean lo mantendré lo más resumido y simple posible pues el objetivo es tener un componente que pueda añadir en varios proyectos, algo genérico.

He decidido hacer el bean como un JLabel e hilo, que cambia su contenido según la hora del sistema.

Proceso

Antes de un JLabel, necesitamos el código que nos devuelva la hora. Para ello, es conveniente contar con los siguientes **atributos**

```
// método útil para formatear horas, da más potencial a Bean
    private SimpleDateFormat sdf = new SimpleDateFormat( pattern: "hh:mm:ss a");

    private String StringCurTime;
    private boolean Format12_24 = true; // control del estado del formato
```

Método para obtener la hora a tiempo real:

```
private void setHour() {
    while (true) {
        StringCurTime = sdf.format( date: Calendar.getInstance().getTime());
        this.setText( text: StringCurTime); // añadir tras extender JLabel
        try {
            Thread.sleep( l: 500);
        } catch (InterruptedException ex) {
        }
    }
}
```

****Incidencia**, este método **no** debe ser llamado directamente en el constructor pues bloqueará la instanciación del mismo.

Este es momento de implementar la interfaz Runnable

```
public class ClockLabel extends JLabel implements Runnable, Serializable  {
```

Entonces, para poder conseguir la hora, llamaremos al método desde un hilo.

```
public void run() {
    setHour();
}
```

Método para cambiar el formateo de la String que visualizará el usuario como hora.

```
public void format12_24() {
    if (Format12_24) {
        sdf = new SimpleDateFormat( pattern: "HH:mm:ss");
        Format12_24 = false;
    } else {
        sdf = new SimpleDateFormat( pattern: "hh:mm:ss a");
        Format12_24 = true;
    }
}
```

Ésta es toda la lógica necesaria para que de manera interna se actualice *StringCurTime*, para visualizarlo, vamos a heredar la clase **JLabel** y implementar la interfaz **Serializable**

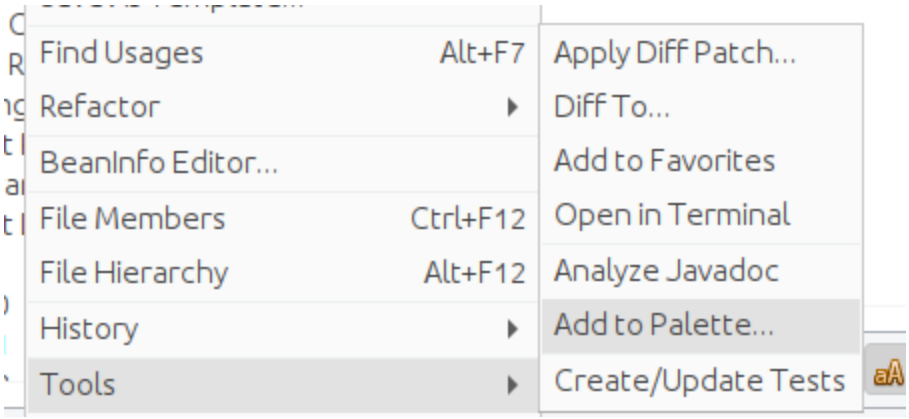
```
public class ClockLabel extends JLabel implements Runnable, Serializable
```

Y, añadiremos en la línea en **setHour()**:

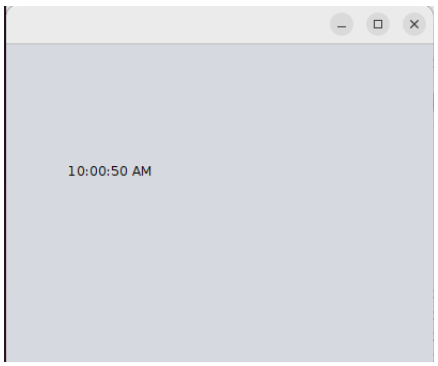
```
this.setText(StringCurTime);
```

Y, lo añadimos a Bean

Click derecho en la clase > herramientas > añadir a la paleta > bean



Añadido el bean a la paleta probarlo en un JFrame



Clase principal - ClockFrame

Dado que ya contamos con el label, nos planteamos como hacer la interfaz que deseamos.
El diseño elegido consta de

Para las fuentes necesitaremos importar al sistema

- ClockLabel → Orbiton:https://fonts.google.com/specimen/Orbitron
- JLabel → Roboto https://fonts.google.com/specimen/Roboto

Cambio de formato

Este botón lo que hará será llamar a una función de ClockLabel que altera su formato horario.

Ciclo del el componente ideado para el componente

Y con cada cambio, el respectivo formato de hora en ClockLabel

Para ello, necesitaremos un atributo booleano para registrar en qué formato nos encontramos.

```
// to control the pm/am
    private boolean fTextLabel;
```

Añadir un JLabel (por razones estéticas se usa en lugar de un JButton).

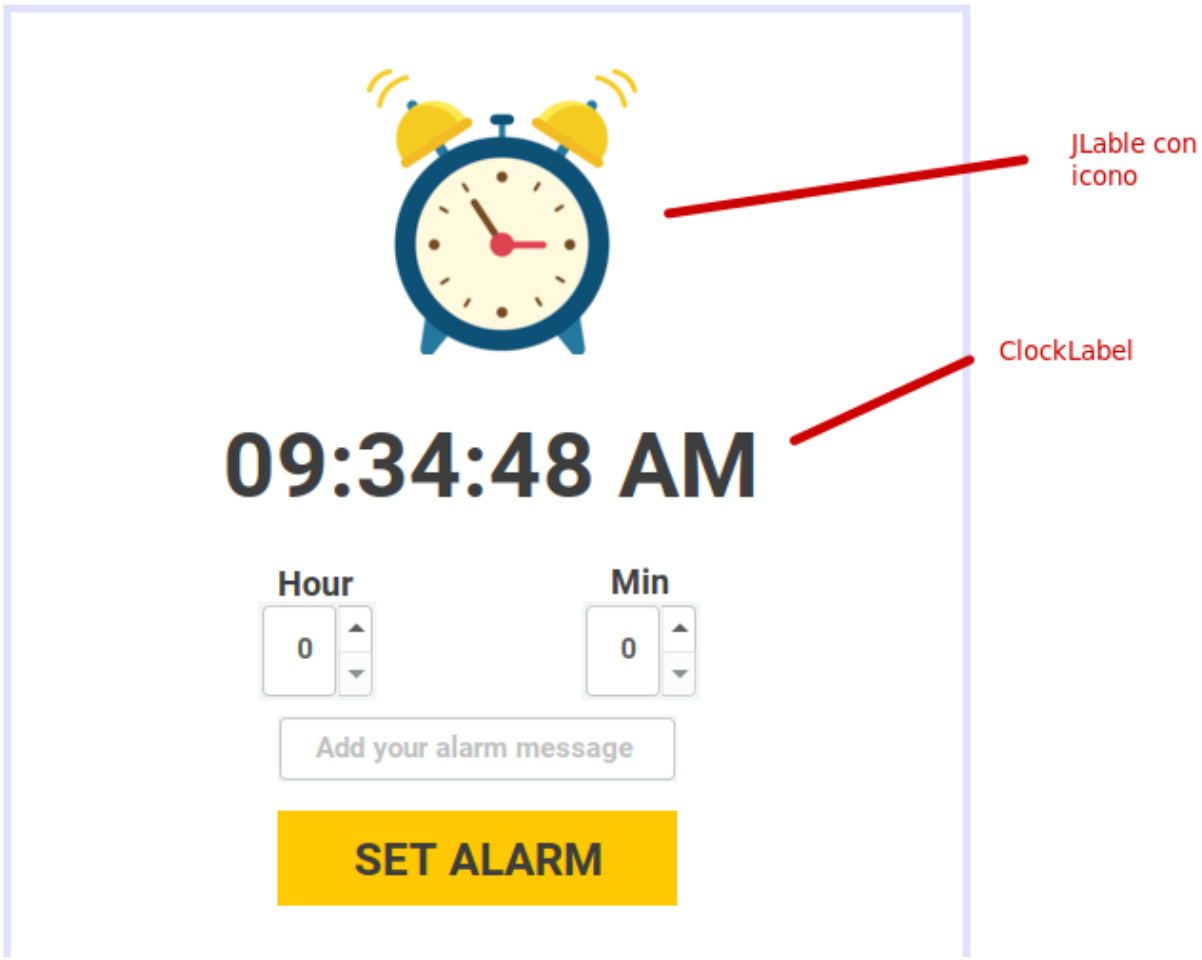
Como valor inicial he optado por poner los dos usos horarios (12h/24h) para indicar de forma visual la función.

lChangeFormat - El componente JLabel que actúa como botón.
Evento onClick en el que:

- Se llama al método que cambia la hora (format12_24).
- Dependiendo de el booleano fTextLabel, se actualiza el texto del botón.

```
private void lChangeFormatMouseClicked(java.awt.event.MouseEvent evt) {
    cLabel.format12_24();
    if (fTextLabel) {
        lChangeFormat.setText( text:"12H");
        fTextLabel = !fTextLabel;
    } else {
        lChangeFormat.setText( text:"24H");
        fTextLabel = !fTextLabel;
    }
    try {
        formatLAlarm();
    } catch (ParseException ex) {
    }
}
```

Clase AlarmDialog



Esta clase será llamada de ClockFrame así que guardaré como atributo un ClockFrame y lo instanciaré en el constructor.

```
public class AlarmDialog extends javax.swing.JDialog {

    ClockFrame parent;

    /**
     * Creates new form dAlarm
     */
    public AlarmDialog(java.awt.Frame parent, boolean modal) {
        super( owner:parent, modal);
        // I add this
        this.parent = (ClockFrame) parent;
        initComponents();
        this.setDefaultCloseOperation( operation:DISPOSE_ON_CLOSE);
```

Dado que deseamos eliminar de la memoria el dialogo una vez enviado los datos, configuraremos la acción de cierre como DISPOSE_ON_CLOSE, asi podremos llamar al método dispose() y olvidarnos del dialog tanto nosotros como la máquina.

```
        this.setDefaultCloseOperation( operation:DISPOSE_ON_CLOSE);
```

Para la conexión a la clase principal ClockFrame, he creado un método público en la clase padre, getAlarmDialogData() que recoge los datos de AlarmDialog una vez rellenados.

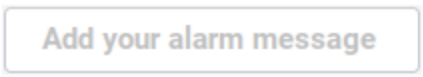
Configuración de la alarma

He optado por usar un dialogo modal hijo de ClockFrame.
Explicaré exclusivamente lo añadido con código y el planteamiento de los componentes.

Lo demás, se configura desde las propiedades de la vista diseño.

Con dos spinner, que recogerán horas de 0-23 en el caso de las horas y, de 0 - 59 en el caso de los minutos.

En el caso del mensaje, nos encontramos ante un JTextField con un **evento onClick** que borrará la pista “Add your alarm message''.



```
    private void lAlarmMsgMouseClicked(java.awt.event.MouseEvent evt) {
        lAlarmMsg.setText( t: "");
    }
```

Finalmente, el JLabel con funciones de evento realiza getters de horas, minutos y mensaje en un evento onClick.


```
    private void btSendMouseClicked(java.awt.event.MouseEvent evt) {
        int hour = (int) spHour.getValue();
        int min = (int) spMin.getValue();
        String msg = lAlarmMsg.getText();

        parent.getAlarmDialogData(hour, min, msg);

        try {
            parent.formatLAlarm();
            parent.OnAlarm();
        } catch (ParseException ex) {}
        dispose();
    }
```

Código añadido en ClockFrame para la conexión

AlarmDialog→ parent.getAlarmDialogData(hour, min, msg);

ClockFrame → Nuevos atributos

```
private int hour, min;
private String msg;
```

Método publico que recibirá la información:

```
public void getAlarmDialogData(int hour, int min, String msg) {
    this.hour = hour;
    this.min = min;
    this.msg=msg;
}
```

AlarmDialog→ parent.formatLAlarm();

ClockFrame → - Dado que recibimos números int en lugar de fechas como lo haciamos

con el ClockLabel, hemos de formatearlos para obtener horas congruentes al formato al que estamos acostumbrados y, en el que ya se muestra ClockLabel .

Ejemplo aclaratorio:
Horas como int → 8:8
Horas reconocibles por humanos → 08:08

Para ello necesitaremos un atributo que guarde esta hora como String.

```
private String alarmString,
```

Y un método que transforme los int hora y minutos, segun el formato que ya se encuentre configurado por format12_24.

```
public void formatLAlarm() throws ParseException {
    String sHora, sMin;
    String result = "";

    if (fTextLabel) {
        sHora = hour < 10 ? "0" + String.valueOf( i:hour) : String.valueOf( i:hour);
        sMin = min < 10 ? "0" + String.valueOf( i:min) : String.valueOf( i:min);
    } else {
        int auxHour = hour;
        if (hour >= 12) {
            auxHour = hour - 12;
        }
        sHora = auxHour < 10 ? "0" + String.valueOf( i:auxHour) : String.valueOf( i:auxHour);
        sMin = min < 10 ? "0" + String.valueOf( i:min) : String.valueOf( i:min);
    }
    result = sHora + ":" + sMin;
    alarmString = result;
    lAlarm.setText( text:result);
}
```

AlarmDialog→ parent.OnAlarm();

ClockFrame → Para implementar la alarma he optado por crear un hilo en el constructor con la función de:

1. Recoger la hora ClockLabel
2. Formatearla a una String para compararla con la AlarmString del paso anterior.

Hilo puente para Strings de ClockLabel con Strings provenientes de int ((AlarmString)

```
public ClockFrame() {
    initComponents();
    //añadido
    new Thread(() -> {
        while (true) {
            try {
                Thread.sleep( l:500);
            } catch (InterruptedException ex) {

            }
            timeString = cLabel.getText().substring( beginIndex:0,  endIndex:5);
        }
    }).start();
```

onAlarm() genera otro hilo que comprueba cada segundo si la hora actual coincide con la de la alarma.
Si es verdadero genera a RingAlarmDialog (que explicaré próximamente).

```
protected void OnAlarm() {
    tAlarm = new Thread(() -> {
        System.out.println( x:"Alarma encendida");

        while (true) {
            //System.out.println("Sonar/actual: "+alarmString+"/"+timeString);
            if (alarmString.equals( anObject:timeString)) {
                RingAlarmDialog ring = new RingAlarmDialog( parent:this,  modal:true);
                ring.setVisible( b:true);
                try {
                    Thread.sleep( l:60000);
                } catch (InterruptedException ex) {
                }
            }
            try {
                Thread.sleep( l:500);
            } catch (InterruptedException ex) {}
        }
    });
    tAlarm.start();

}
```

Ya que iniciamos la alarma hemos de apagarla.
Lo haremos desde el método **OffAlarm**() que será escuchado por el radio button rbOff.

```
private void rbOffMouseClicked(java.awt.event.MouseEvent evt)
    lAlarm.setVisible( aFlag: false);
    lEnun.setVisible( aFlag: false);
    OffAlarm();
```

Como detalle, añadiremos un label **lAlarm** que mostrará la String formateada anteriormente explicada en formatAlarm().

Y, lo incorporaremos al evento **lChangeFormatMouseClicked** para convertir la hora de lAlarm a la vez que la hora del cLabel (ClockLabel)

```
private void lChangeFormatMouseClicked(java.awt.event.Mous
    cLabel.format12_24();
    if (fTextLabel) {
        lChangeFormat.setText( text: "12H");
        fTextLabel = !fTextLabel;
    } else {
        lChangeFormat.setText( text: "24H");
        fTextLabel = !fTextLabel;
    }
    try {
        formatLAlarm();
    } catch (ParseException ex) {
    }
}
```

Clase AlarmDialogRing

Dialogo casi enteramente visual con un constructor sobreescrito con una variable msg

```
private ClockFrame parent=(ClockFrame) this.getParent();
private String msg=parent.getMsg();

public RingAlarmDialog(java.awt.Frame parent, boolean modal) {
    super( owner:parent, modal);
    initComponents();
    lAlarmText.setText( text:msg);

}
```

que se muestra en un label lAlarmText

El campo vacío se controla dos saltos de clase antes. Instanciandose como “”;