



IES VALLE DEL JERTE PLASENCIA

PRIMERA PRÁCTICA UNIDAD 3 1ºDAM

# DESARROLLO DE INTERFACES SWING

*Abel López Ruiz*

Editado por



22 de diciembre de 2024

# ÍNDICE GENERAL

---

<b>1. Introducción</b>	<b>2</b>
1.1. Interfaces . . . . .	2
1.2. Interfaces en Java . . . . .	3
<b>2. Swing en Java</b>	<b>6</b>
2.1. Swing en Eclipse . . . . .	6
2.2. Swing en NetBeans . . . . .	15
<b>3. Práctica con Java Swing</b>	<b>19</b>
3.1. Práctica Interfaz . . . . .	19
3.2. Prácticas Interfaces propuestas . . . . .	29
<b>4. Interfaces en Java</b>	<b>31</b>
4.1. ¿Qué son los interfaces en Java? . . . . .	31
4.2. Ejercicios propuestos interfaces . . . . .	34
<b>5. Entrega</b>	<b>35</b>
<b>6. Bibliografía</b>	<b>36</b>

# 1. Introducción

## 1.1. Interfaces

Antes de comenzar esta práctica, se puede hacer una breve introducción para que son tan necesarias las interfaces y qué son. Cualquier programa que se desarrolle en cualquier IDE necesita un interfaz (ojo no confundir con los interfaces de una clase que es algo parecido y comentaremos en clase) donde el usuario interactúe con el programa para cualquier cosa, ya sea para introducir un valor por teclado, pulsar un botón, etc. Por lo que es necesario que la interfaz sea lo más intuitivo posible y de sencillo uso.

Por lo que cualquier máquina o programa desarrollado necesita de un interfaz por muy básico que sea para que el humano interactúe con dicha máquina o programa. En el ámbito del desarrollo de software es muy típico implementar en muchas de los proyectos que un cliente solicita a una empresa interfaces gráficas de usuarios, muy intuitivas de usar, y que no requieran conocimientos avanzados para el usuario final para su uso, estas son conocidas como GUI (Graphic User Interface).

Una de las interfaces más intuitivas que se pueden crear y que serían muy útiles y fáciles para el usuario final es la ventana con una serie de componentes muy intuitivos con el que el usuario mediante el teclado o el ratón pueda interactuar de una forma sencilla. Esto es una interfaz mucho más comprensible para el usuario inexperto que una línea de comando donde para interactuar con el programa necesita introducir comandos y parámetros.

En las interfaces existen unos estándares de calidad, que mide como de bueno es una interfaz, las interfaces de calidad deben reunir una serie de propiedades: claridad, deben ser concisas(no llena de muchos botones por ejemplo, debe ser simple), familiar, tener consistencia, estética, eficiencia, capacidad de respuesta entre otras.

Peter Morville creó un modelo de criterios del diseño de una interfaz como se muestra en la Figura 1.1:

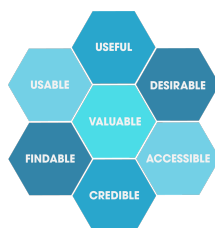


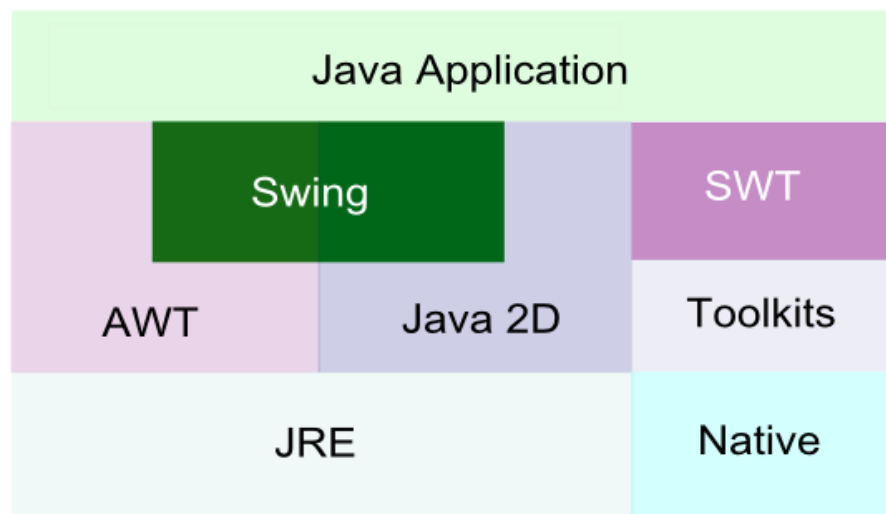
Figura 1.1: Criterios diseño Interfaces

## 1.2. Interfaces en Java

Las bibliotecas que podemos usar en Java para la creación de interfaces gráficos para nuestros programas son muchos, pero los más conocidos son AWT, Swing, y otras herramientas como SWT. También hay muchas otras que se pueden utilizar menos conocidas. Pero en esta práctica se va a utilizar las librerías Swing de Java para implementar interfaces gráficas.

Algunas diferencias entre AWT (Abstract Windows Toolkit) que ya quedó obsoleto y depende del sistema operativo (en cada sistema operativo se verá la ventana distinta y habrá determinadas herramientas del sistema operativo que no podremos usar) y Swing, es que Swing es más portable, no depende del sistema nativo de ventanas del sistema operativo y es una expansión de AWT que muchas de sus funciones quedaron obsoletas. En AWT no se pueden hacer cambios en el aspecto de los componentes según el sistema operativo. La librería AWT se encuentra en `java.AWT`, mientras que `java Swing` se encuentra en `javax.swing`.

Las clases de Java Swing suelen llevar una J delante como por ejemplo el famoso `JButton`, esto no ocurre en las clases de AWT, que en este caso sería `Button`. El SWT es una alternativa en eclipse a AWT y Java Swing que se puede instalar desde eclipse en la ventana de help o ayuda. En la siguiente figura se tiene los tipos de librerías de interfaces de Java y sus capas:



### Java Graphics - The Layer Cake

Figura 1.2: Capas de AWT, 2D, Java Swing, SWT.

Como nos centraremos en la biblioteca Java Swing, en la siguiente figura se muestra la jerarquía de clases de la biblioteca Java Swing. Tanto de todos los componentes como de la JFrame que es el objeto ventana clásico de Swing:

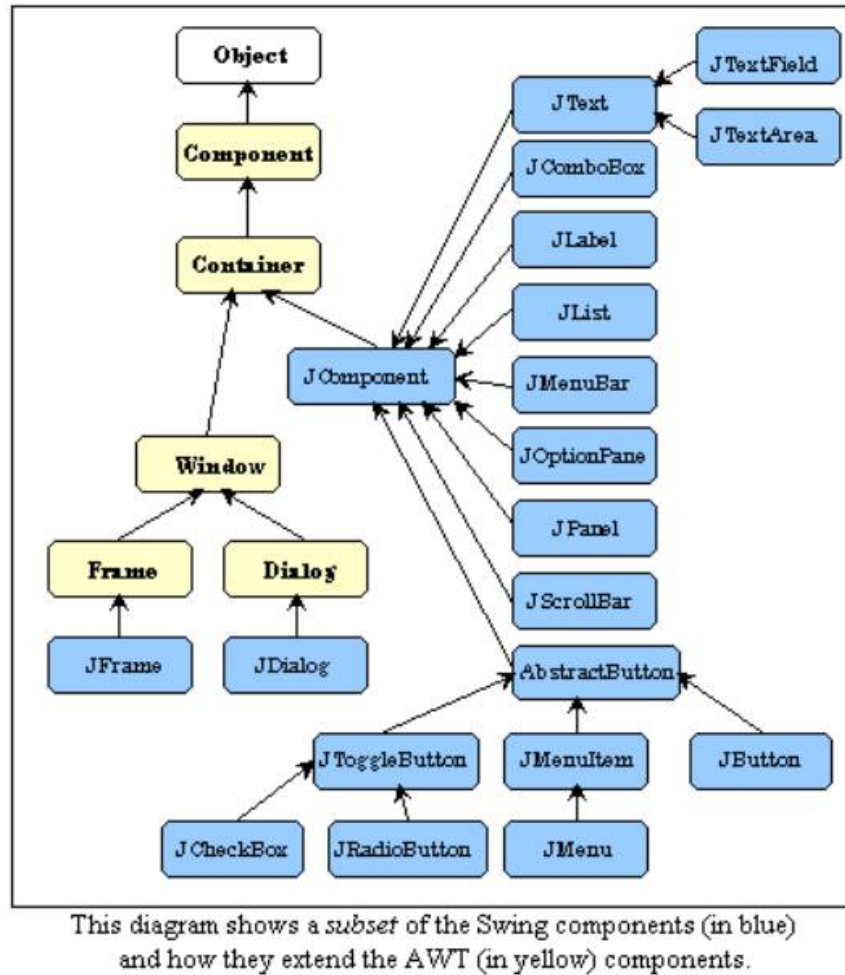


Figura 1.3: Jerarquía Java Swing

Por lo que es posible modelar los objetos de estas librerías como una serie de capas como se muestra en las siguientes figuras:

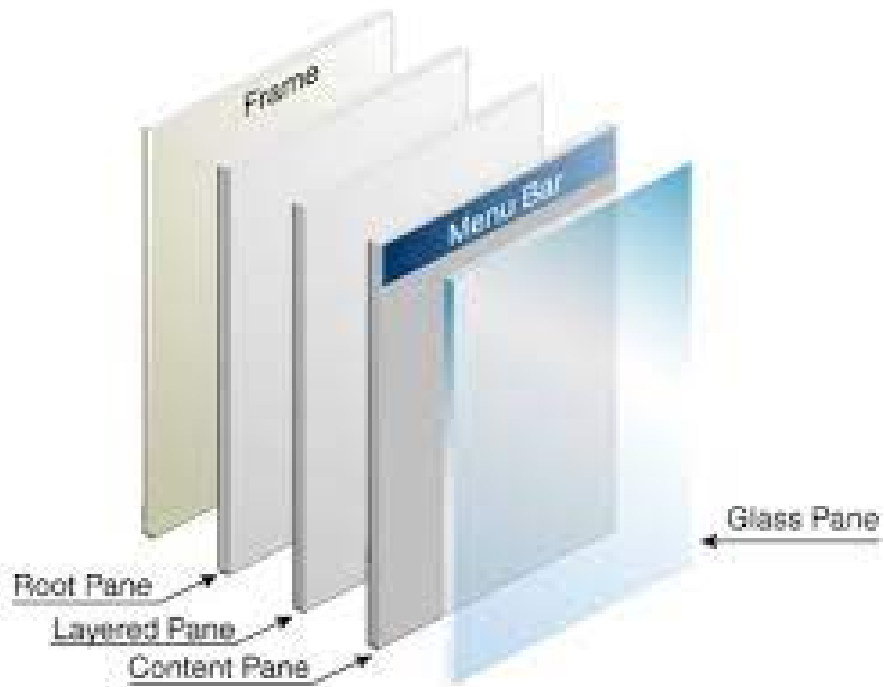
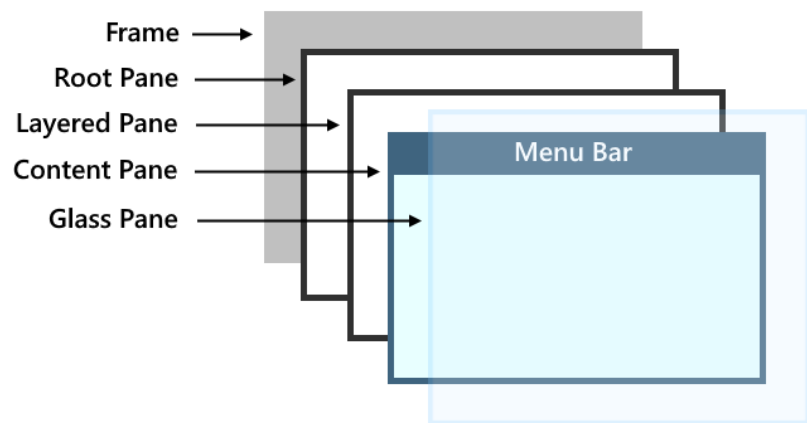


Figura 1.4: Modelo de capas.



©Buzzword Inc.

Figura 1.5: Modelo de capas.

## 2. Swing en Java

### 2.1. Swing en Eclipse

En esta parte se va a trabajar con Swing en Eclipse, se van a mostrar dos formas de trabajar con Swing en Eclipse, una de ellas es programando todos los componentes mediante código fuente y otras es con una herramienta gráfica donde se pueden trabajar directamente sobre la ventana y los componentes de Swing simplemente arrastrando los componentes con el ratón en la ventana o Frame.

Antes de nada se va a crear un proyecto con una clase, como siempre se ha hecho y se va a colocar el import `javax.swing.*` para importar las clases de java Swing (si además hace falta crear una clase y un paquete se pueden crear fácilmente):

```
1 package venta1;
2
3
4 import javax.swing.*;
5
6 public class ven {
7
8
9     public static void main(String args[])
10    {
11
12
13
14
15    }
16
17
18
19 }
20
```

Figura 2.1: Uso de las clases Swing en Java.

Además, para crear una primera ventana, es una instancia típica de Swing, es la conocida JFrame, el objeto se creará de la siguiente forma:

```
import javax.swing.*;

public class ven {

    public static void main(String args[])
    {

        JFrame ventana= new JFrame("Nueva venbtana");
        ventana.setBounds(100,100,400,400);
        ventana.setResizable(false);
        ventana.setLayout(null);
        ventana.setVisible(true);

    }

}
```

Figura 2.2: Creación de mi primera ventana.

Cuando se compila en Eclipse se obtiene algo como esto:

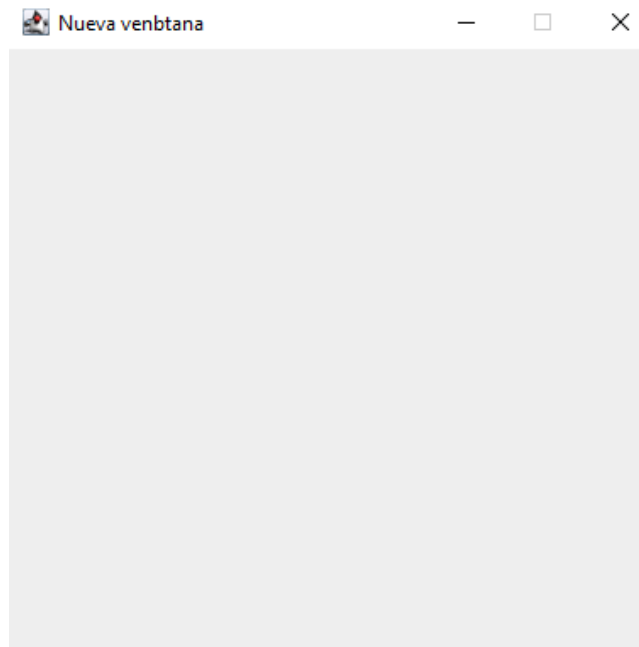


Figura 2.3: Creación de mi primera ventana.

Una vez creada, puedo trabajar sobre mi ventana desde una herramienta de Swing que me permite trabajar y añadir componentes a mi ventana, para acceder a esta herramienta pinchamos con click derecho en nuestra clase creada como se muestra a continuación:



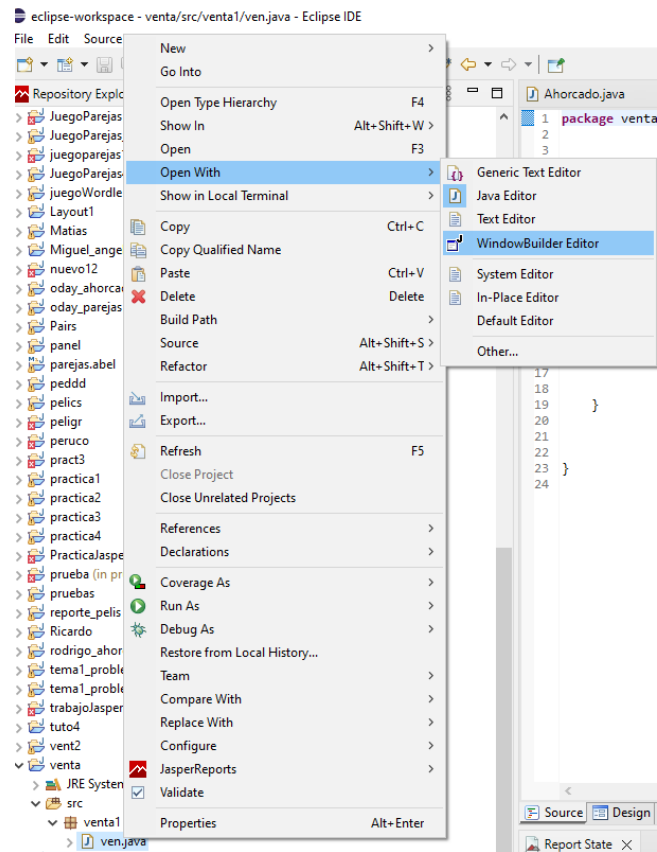


Figura 2.4: Editor de ventanas.

Después de hacer esto, tenemos que nos aparecen las siguientes pestañas, una para el código, y otra para el editor de ventanas:

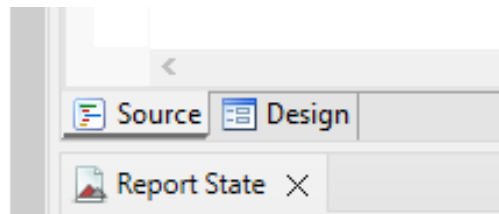


Figura 2.5: Editor de ventanas.

Si pulsamos en el tab o pestaña tab se va abrir la siguiente ventana:

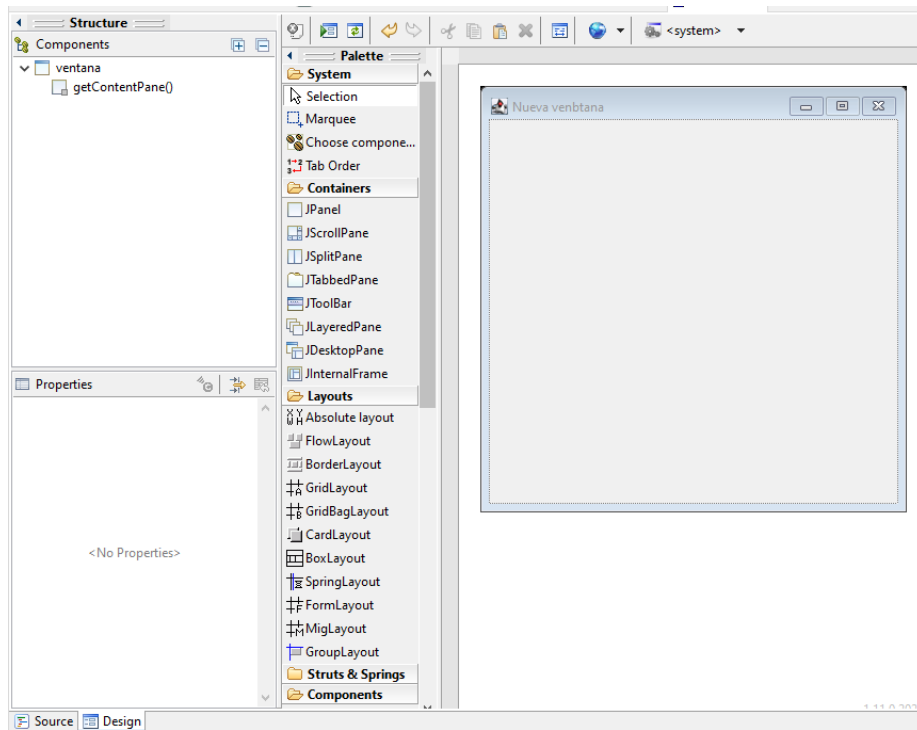


Figura 2.6: Editor de ventanas.

Antes de nada, se detallarán algunas aclaraciones, aunque no se exige al alumno que sepan todo sobre las clases de swing, esto es una práctica de introducción y conocimiento de esta herramienta de interfaces. Pero antes de nada, comentar que los métodos `getContentPane()` acceden al contenido de la ventana, del panel, Layout nos indica como se van a distribuir los elementos de la ventana, si va a hacer una rejilla, `BorderLayout` (dividido en los puntos cardinales) pero nosotros trabajaremos con un layout absoluto de coordenadas x e y.

Otro aspecto importante a tener en cuenta es el conjunto de ventanas que aparecen, una de ellas es la ventana de componentes donde aparecerán todos los componentes como los `JButton`, o los `TextField` que vamos agregando a la ventana, en la ventana de propiedades se podrá modificar los valores por defectos de cada componente al marcarlo con el ratón, como el color, el texto de un botón, incluso hay una pestaña de los eventos.

Un evento es algo que ocurre y el componente se comporta de una cierta forma realizando algo, por ejemplo los eventos de un botón, puede ser pulsarlo, hacer doble click, pasar el ratón por encima, o pulsar una cierta tecla, u otros muchos. Más adelante en la práctica se van a ver un ejemplo de ello.

Veamos un ejemplo de como cambiar el contenido de una JFrame, al marcarlo nos aparecen las propiedades y cambiamos el color de fondo o background:

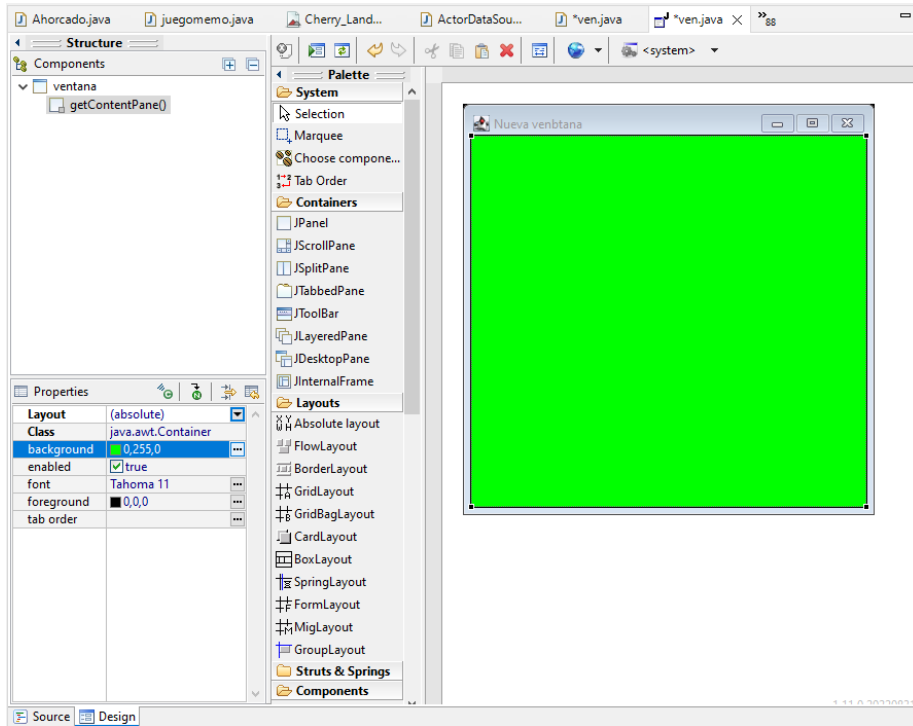


Figura 2.7: Editor de ventanas. Cambio de propiedades.

Se puede observar en la figura anterior que hay una pestaña para cambiar el layout. Luego repasaremos un poco sobre este tema. Antes de nada vamos añadir un botón y en la ventana palette se busca en componentes, y lo arrastramos sobre la ventana, luego lo marcamos con el ratón para ver sus propiedades.

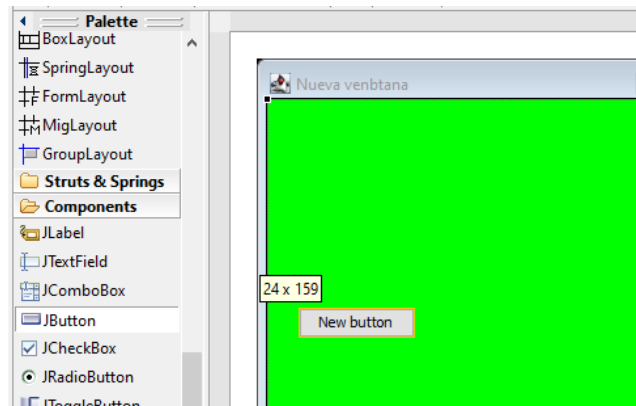


Figura 2.8: Editor de ventanas. Botón.

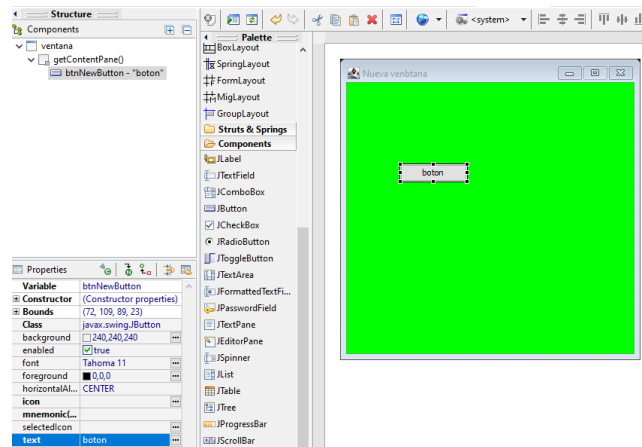


Figura 2.9: Propiedades Botón.

En la figura anterior se ha podido observar como se ha cambiado una de las propiedades del botón, en este caso texto, además vamos a pinchar en la pestaña fuente y se podrá comprobar que hay un código sobre la creación de un objeto botón con un nombre asignado por defecto y añadido con el método add, todo esto generado de forma automática:

```
1 package ventan1;  
2  
3  
4 import javax.swing.*;  
5 import java.awt.Color;  
6  
7 public class ven {  
8  
9  
10 public static void main(String args[])  
11 {  
12  
13     JFrame ventana= new JFrame("Nueva venbtana");  
14     ventana.getContentPane().setBackground(new Color(0, 255, 0));  
15     ventana.setBounds(100,100,400,400);  
16     ventana.setResizable(false);  
17     ventana.getContentPane().setLayout(null);  
18  
19     JButton btnNewButton = new JButton("boton");  
20     btnNewButton.setBounds(72, 109, 89, 23);  
21     ventana.getContentPane().add(btnNewButton);  
22     ventana.setVisible(true);  
23  
24 }  
25  
26 }  
27 }
```

Figura 2.10: Código generado.

Además, también debe fijarse que si se necesita alguna otra API de java o librería estándar, como `awt.Color`, lo importa automáticamente. Compruebe el código que se añadió manualmente con el generado automáticamente respecto a la Figura 2.2.

Veamos ahora algún ejemplo de layout, por ejemplo en el caso anterior era Absolute Layout, se puede colocar los componentes en cualquier coordenada (x,y) de la ventana,

otro ejemplo aunque podéis comprobar otro es el BorderLayout, veamos un ejemplo de añadir varios botones (volvemos a pinchar en el contenido de la ventana y cambiamos su layout (área de dibujado)):

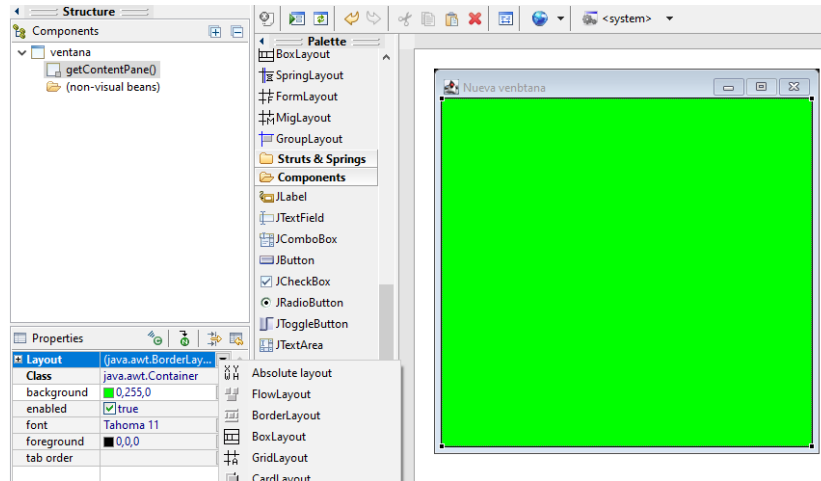


Figura 2.11: Layout de ventana.

Este tipo de Layout nos permite añadir elementos, en centro, este, oeste, norte, etc. Lo podemos contemplar en la Figura siguiente:

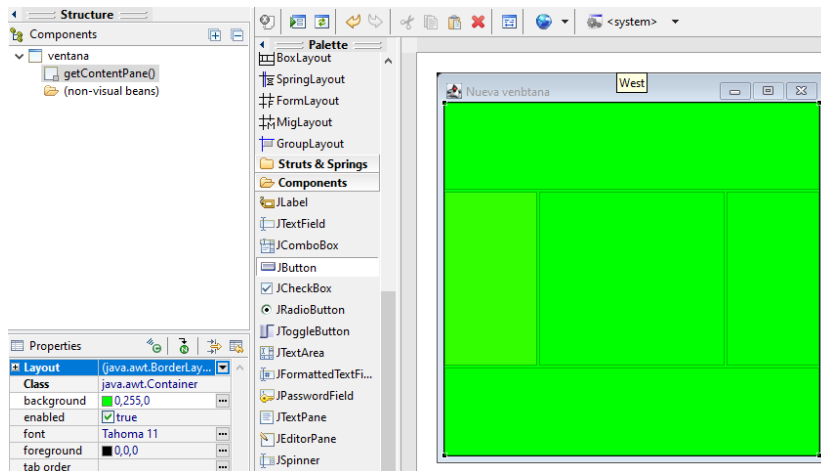


Figura 2.12: Layout de ventana. BorderLayout

Se van a agregar varios botones en cada una de las posiciones para completar el ejemplo, aunque mayormente solemos trabajar con Layout absoluto, también existen Layouts de rejillas o grid, que te dividen el área de trabajo en una cuadrícula, también se mostrará una imagen:

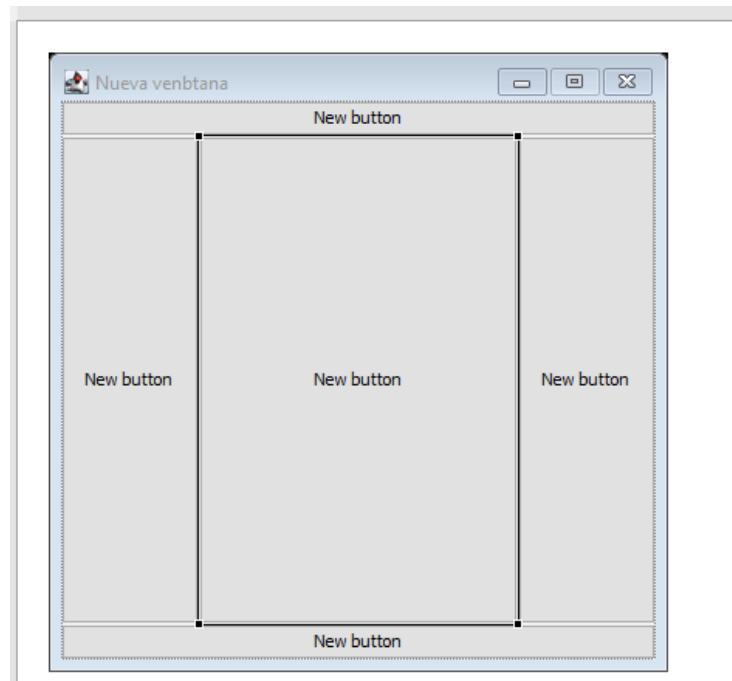


Figura 2.13: Layout de ventana. BorderLayout

Con una rejilla:

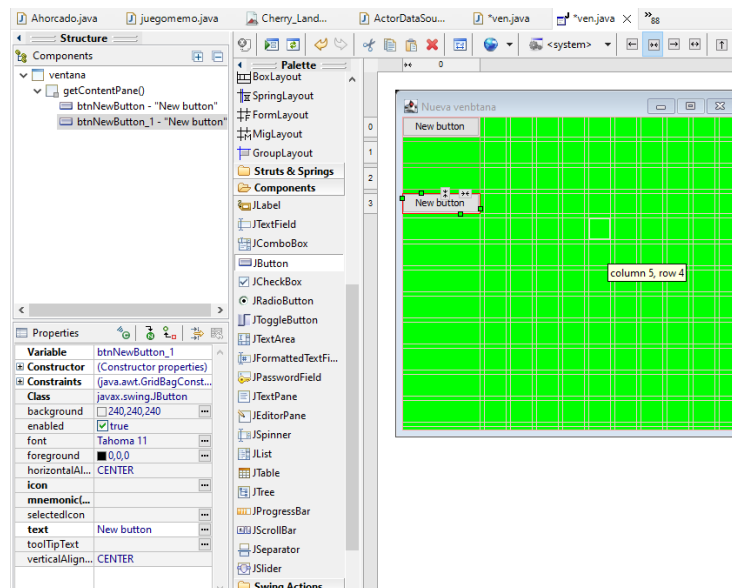


Figura 2.14: Layout Grid

En cuanto a los eventos se activan en una de las pestañas de propiedades, por ejemplo cuando es clickeado un botón:

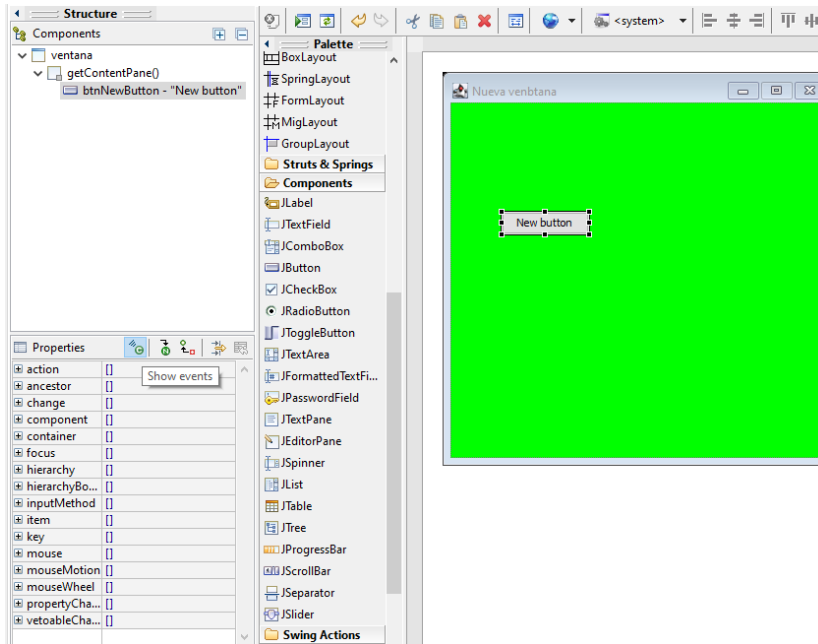


Figura 2.15: Eventos

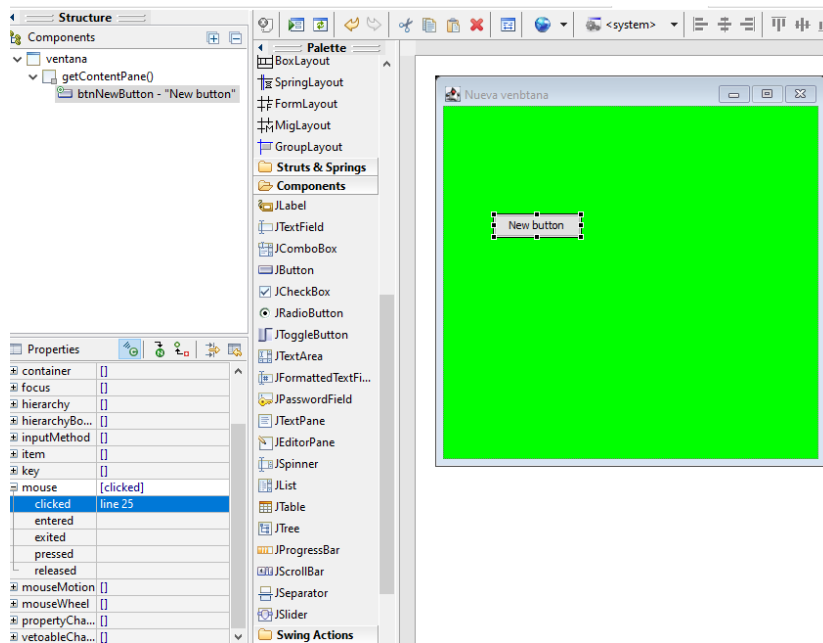


Figura 2.16: Eventos

En la ventana anterior, si nos vamos a mouse y luego en clicked pinchamos dos veces nos sale el método para el evento, lo que es el esqueleto, algo así en la pestaña source:

```
package ventan1;

import javax.swing.*;
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class ven {

    public static void main(String args[])
    {

        JFrame ventana= new JFrame("Nueva venbtana");
        ventana.getContentPane().setBackground(new Color(0, 255, 0));
        ventana.getContentPane().setLayout(null);

        JButton btnNewButton = new JButton("New button");
        btnNewButton.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
            }
        });
        btnNewButton.setBounds(52, 111, 89, 23);
        ventana.getContentPane().add(btnNewButton);
        ventana.setBounds(100,100,400,400);
        ventana.setResizable(false);
        ventana.setVisible(true);

    }
}
```

Figura 2.17: Eventos, código generado.

## 2.2. Swing en NetBeans

Ya se ha trabajado con Swing con Eclipse, pero también se va a mostrar al menos de forma escueta como se trabaja swing en Netbeans, aunque la forma es muy similar, y todos los componentes son lo mismo.

Se insta al alumno a realizar las mismas práctica en los IDE's para practicar en los dos y mejorar sus habilidades en los dos entornos.

Antes de nada comentar que los componentes de swing, son JavaBeans, ¿qué es un JavaBean o Bean?. Se le deja como propuesta al alumnos investigar sobre lo que es y



qué relación tiene que Swing, pero un Bean es un componente que tiene propiedades, métodos set y get, eventos, etc. Esto se define como un software reutilizable y que es muy útil, en este tipo de software se basa swing.

Primeramente para trabajar en Netbeans con Swing, se va a crear un proyecto con el nombre que deseéis como se ha realizado siempre. Se borra del paquete el archivo .java por defecto que es una clase. Le damos sobre el paquete-nuevo-JFrame Form.

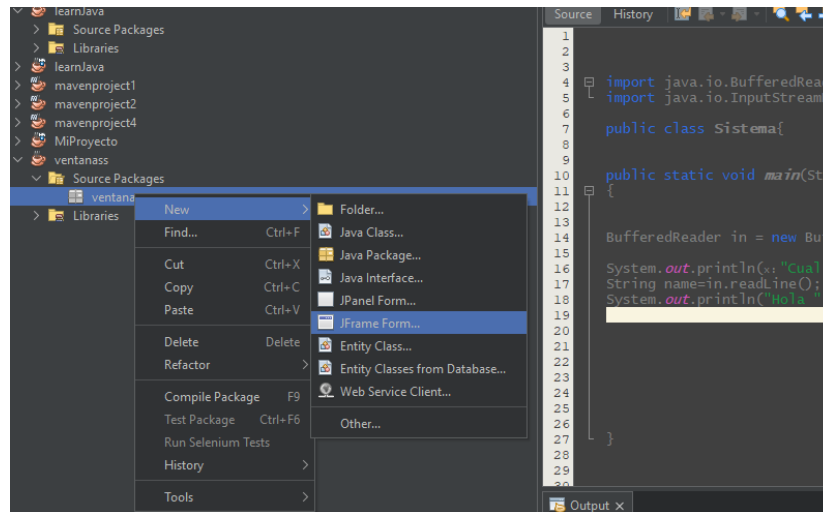


Figura 2.18: Proyecto swing en Netbeans

Se puede comprobar que sale un poco distinto que Eclipse, de hecho en la parte de código aparece lo siguiente:

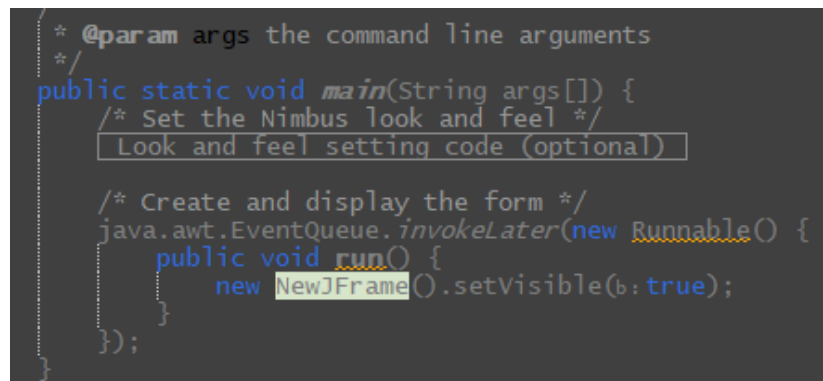


Figura 2.19: Hilo de ejecución.

Ahora explicaremos esta parte, que se puede borrar si se quiere, está dentro del main, y pone que es de la clase EventQueue (cola de eventos), mientras tanto la pestaña de design o diseño es así:

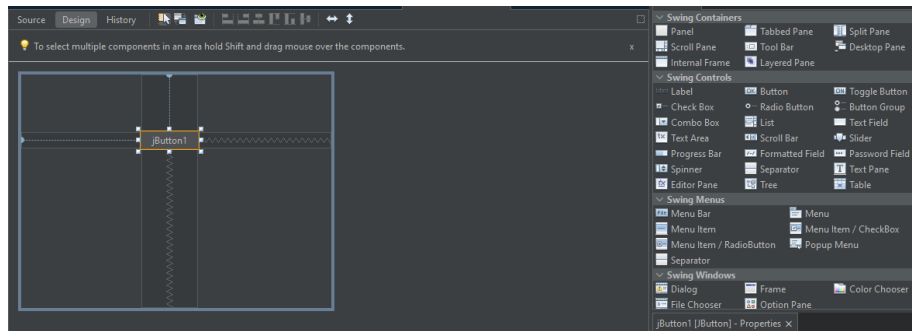
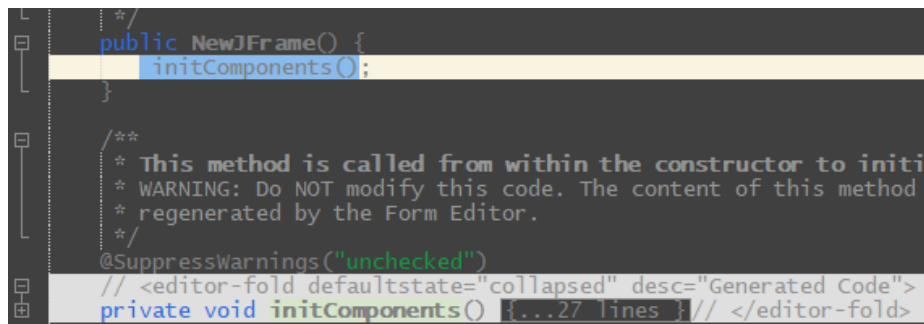


Figura 2.20: Pestaña de diseño en Netbeans.

Respecto al código anterior que incluye por defecto Netbeans sobre un método `run`, es un hilo de ejecución paralelo o independiente que está constantemente a la escucha de eventos, de tal forma que los eventos se ejecutarán en ese hilo de ejecución y el interfaz estará en un evento distinto de ejecución, ¿para qué se crea este hilo de ejecución para eventos?, pues porque si se realizan eventos que tardan mucho tiempo en realizar o de difícil cálculo al realizarse en el mismo hilo de ejecución paralizaría la ventana o interfaz hasta que termine el evento. Con esto se le invita al alumno a investigar qué es un hilo en java o `thread` y qué utilidad tiene.

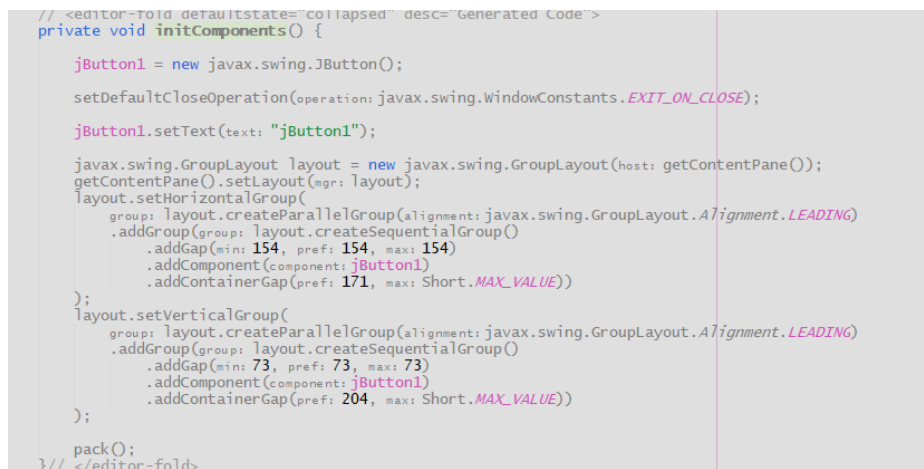
Otra cosa importante es que este código no se genera en Eclipse de forma automática como en Netbeans, pero cabe preguntar si se puede añadir, la respuesta es que sí, pero deberás agregarlo manualmente, o crear una plantilla (template) que cada vez que pinches nueva `JFrame`, te lo genere de forma automática.

Otra cosa que comentar es el método que en realidad es un constructor  `initComponents()`, al que se le puede cambiar el nombre y viene todo contraído y se puede ver dándole al `+` y aparecerá todo el código de mi ventana y los componentes que voy añadiendo en la parte de diseño, todo el código se va generando ahí y te construye la ventana como un objeto con todo lo que le pongas, y luego en el hilo de eventos lo único que hace es llamar al constructor `new JFrame()` llamando a la vez al método para hacerla visible. Este código de `init` no ocurre como en Eclipse que se podía modificar, en Netbeans es diferente y no se puede modificar, solo lo deja modificar desde la pestaña `design`, solo se puede modificar ciertas cosas como los métodos de los eventos y nombres de los componentes.



```
public NewJFrame() {  
    initComponents();  
}  
  
/**  
 * This method is called from within the constructor to initi  
 * WARNING: Do NOT modify this code. The content of this method  
 * regenerated by the Form Editor.  
 */  
@SuppressWarnings("unchecked")  
// <editor-fold defaultstate="collapsed" desc="Generated Code">  
private void initComponents() {...
```

Figura 2.21: Método initComponents.



```
// <editor-fold defaultstate="collapsed" desc="Generated Code">  
private void initComponents() {  
  
    jButton1 = new javax.swing.JButton();  
  
    setDefaultCloseOperation(operation: javax.swing.WindowConstants.EXIT_ON_CLOSE);  
  
    jButton1.setText(text: "jButton1");  
  
    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(host: getContentPane());  
    getContentPane().setLayout(mgr: layout);  
    layout.setHorizontalGroup(  
        group: layout.createParallelGroup(alignment: javax.swing.GroupLayout.Alignment.LEADING)  
        .addGroup(group: layout.createSequentialGroup())  
        .addGap(min: 154, pref: 154, max: 154)  
        .addComponent(component: jButton1)  
        .addContainerGap(pref: 171, max: Short.MAX_VALUE)  
    );  
    layout.setVerticalGroup(  
        group: layout.createParallelGroup(alignment: javax.swing.GroupLayout.Alignment.LEADING)  
        .addGroup(group: layout.createSequentialGroup())  
        .addGap(min: 73, pref: 73, max: 73)  
        .addComponent(component: jButton1)  
        .addContainerGap(pref: 204, max: Short.MAX_VALUE)  
    );  
  
    pack();  
} // </editor-fold>
```

Figura 2.22: Método initComponents desplegado.

También se le propone al alumno investigar qué tipo de función es `pack()` en Swing. Se puede comprobar en el anterior código que aparece el código que he agregado. Os podéis fijar que en la Figura 2.21 lo que hace el constructor de ventanas `NewFrame` es llamar al `initComponents`. Otra cosa interesante es que aparece una especie de `@SuppressWarnings`, eso que lleva una arroba es lo que se conoce en Java como una anotación, se invita al alumno a estudiar que es una anotación en Java para aprender más sobre ello. Este tipo de anotación hace que cuando compile el compilador me ignore los warnings o advertencias y no me salgan.

Con todo lo expuesto hasta aquí el alumno debería de ser capaz de trabajar con swing tanto en Netbeans como en Eclipse, espero que sea entretenido la práctica que se va a realizar y además aprendáis mucho más sobre el manejo de estos dos IDE's. En el próximo capítulo de va a proponer una práctica sencilla sobre Swing.

## 3. Práctica con Java Swing

### 3.1. Práctica Interfaz

En un proyecto de Java, los archivos de los que se componen no son siempre código fuente, es decir, archivos .java. En un proyecto se puede utilizar ficheros de formato imagen .jpg o .png o archivos de texto, etc. Es decir un proyecto puede estar compuesto de muchos tipos de ficheros según los necesitemos, además estos archivos pueden ir almacenados en carpetas dentro del proyecto, una carpeta para imágenes, otra para texto, otra para vídeo, otra para audio, etc. Veamos como en la siguiente imagen en el proyecto se ha incluido una imagen de extensión .jpg:

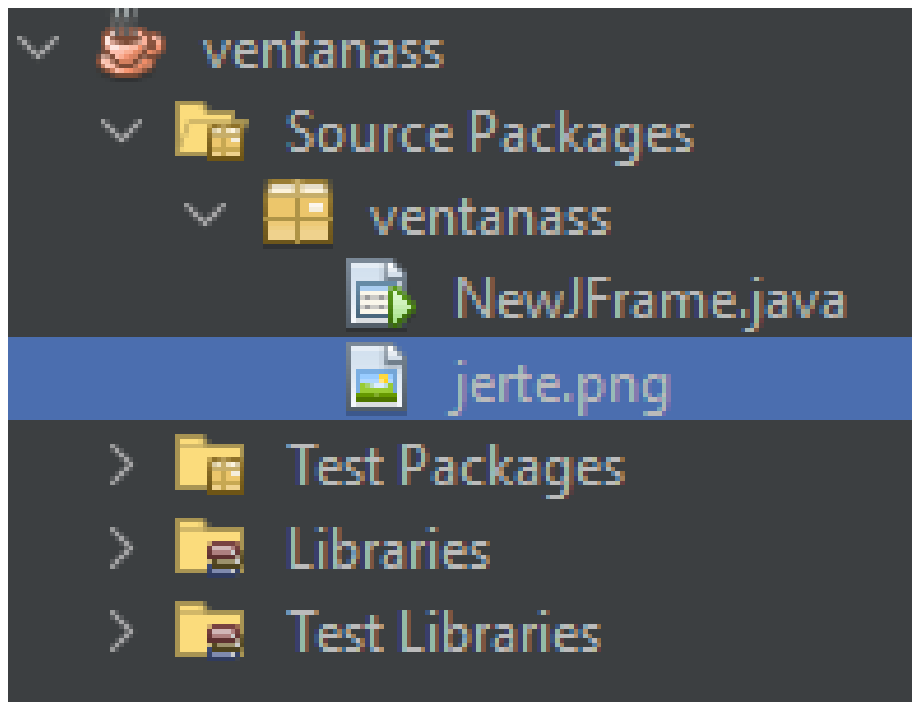


Figura 2.23: Ficheros proyectos.

Además, puedo crear muchísimas clases y usarlas en mi proyecto.

Una primera pequeña práctica que se va a hacer es cambiar el icono de una ventana y para ello se usa el método `setIcon`, y dentro vamos a meter un objeto de tipo `Image`, el cual se creará con `new`. Veamos pues en la siguiente imagen como se puede hacer en Netbeans:

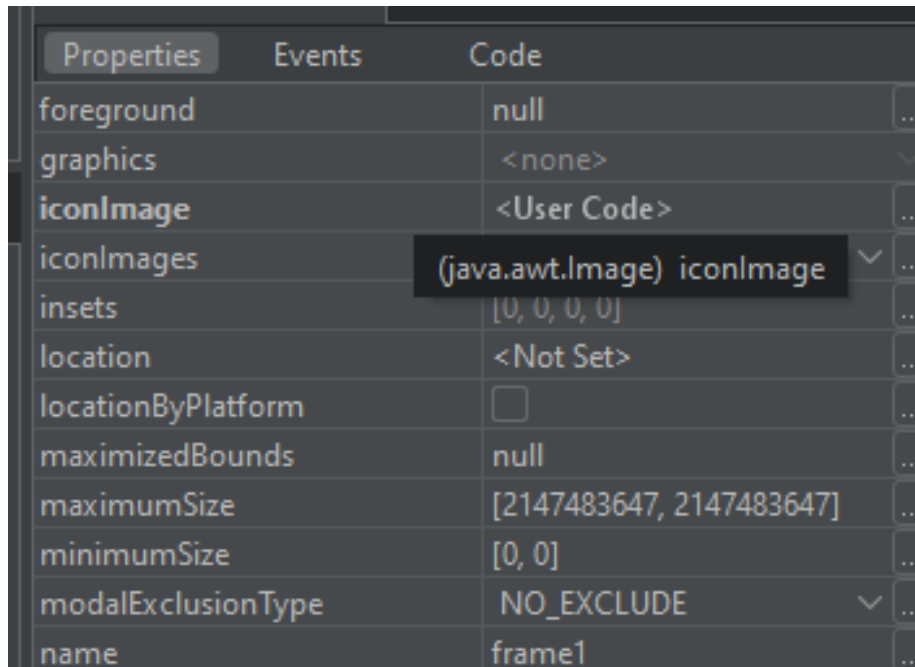


Figura 2.24: Icono

Como se puede ver en la imagen anterior en la propiedad de la Frame iconImage, pulsamos dos veces, vemos que nos sale un popup (o ventana emergente) que aparece la clase de iconImage conocida por java.awt.Image. Pinchamos dos veces y ponemos el siguiente código para incluir una imagen:

```
new ImageIcon(this.getClass().getResource("/ventanass/jerte.png")).getImage()
```

Lo incluimos como se muestra en la imagen en la ventana de diálogo que nos aparece de la siguiente forma en custom code:

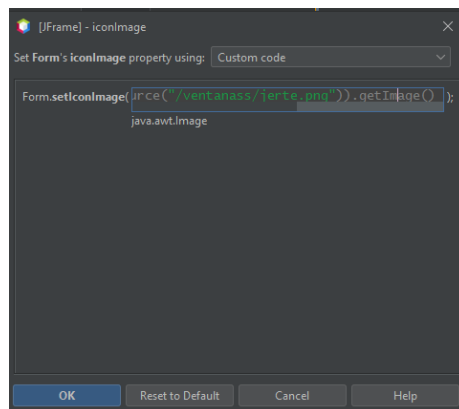


Figura 2.25: Icono

Una vez le hayamos dado a Ok, comprobamos en la pestaña source que aparece añadido en el código como sigue en el initComponents():

```
setDefaultCloseOperation(operation: javax.swing.WindowConstants.EXIT_ON_CLOSE);  
setIconImage( image: new ImageIcon(location: this.getClass().getResource(name: "/ventanass/jerte.png")).getImage());
```

Figura 2.26: Icono

Nos aparecerá una ventana con la siguiente forma:

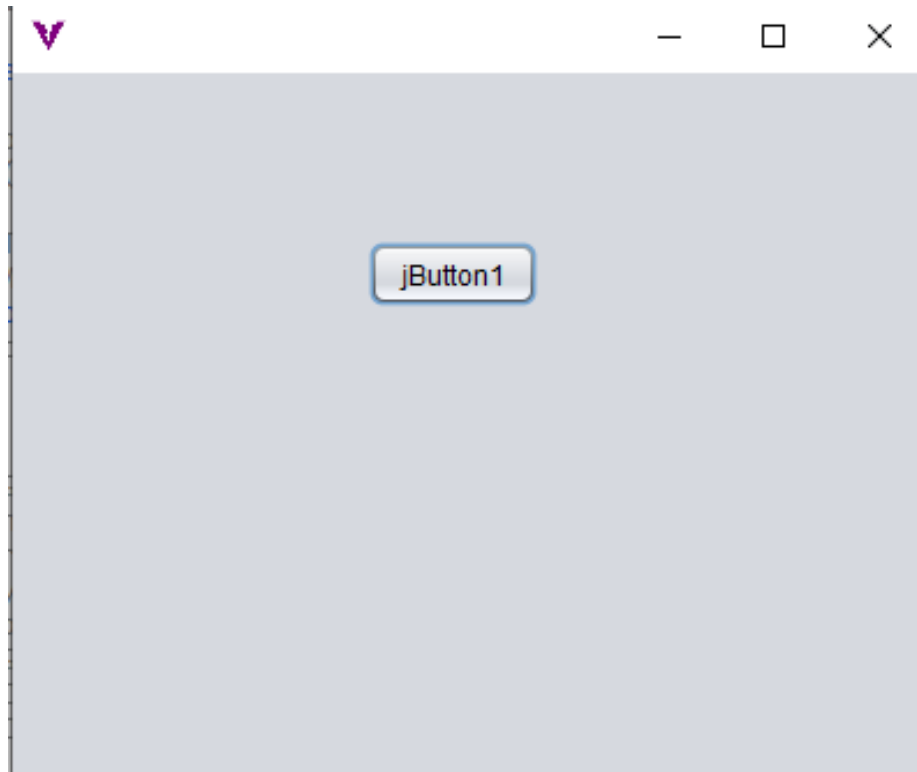


Figura 2.27: Vantana con Icono

El / indica la carpeta source del proyecto, y ventanass el paquete y luego el fichero imagen.

En lo siguiente el alumno realizará una práctica guiada para que domine el uso de estas herramientas que se pueden utilizar tanto en Eclipse como en Netbeans, se procederá a hacerla en Netbeans pero se le invita al alumno a hacerlas también en Eclipse ya que es el mismo programa.

Primeramente el alumno cambiará el título de la ventana con el nombre "Calificaciones nombre alumno" y además añadirá un JLabel que ponga "Calificaciones exámenes nombre alumno" de la siguiente manera:

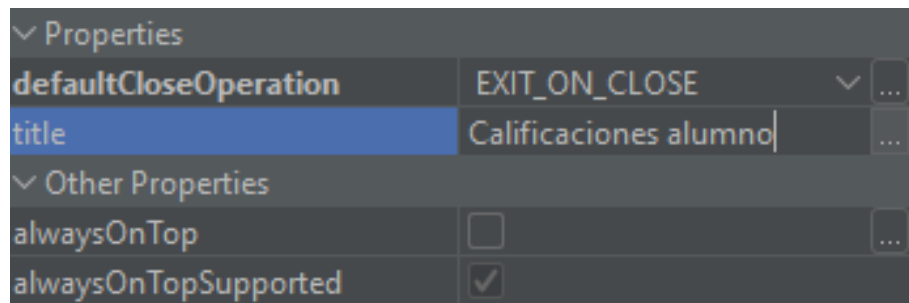


Figura 2.28: Título ventana

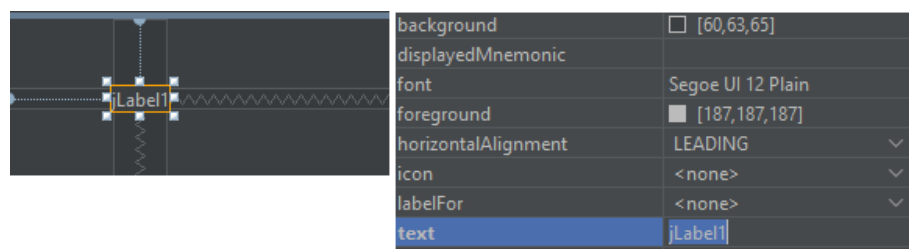


Figura 2.29: JLabel ventana

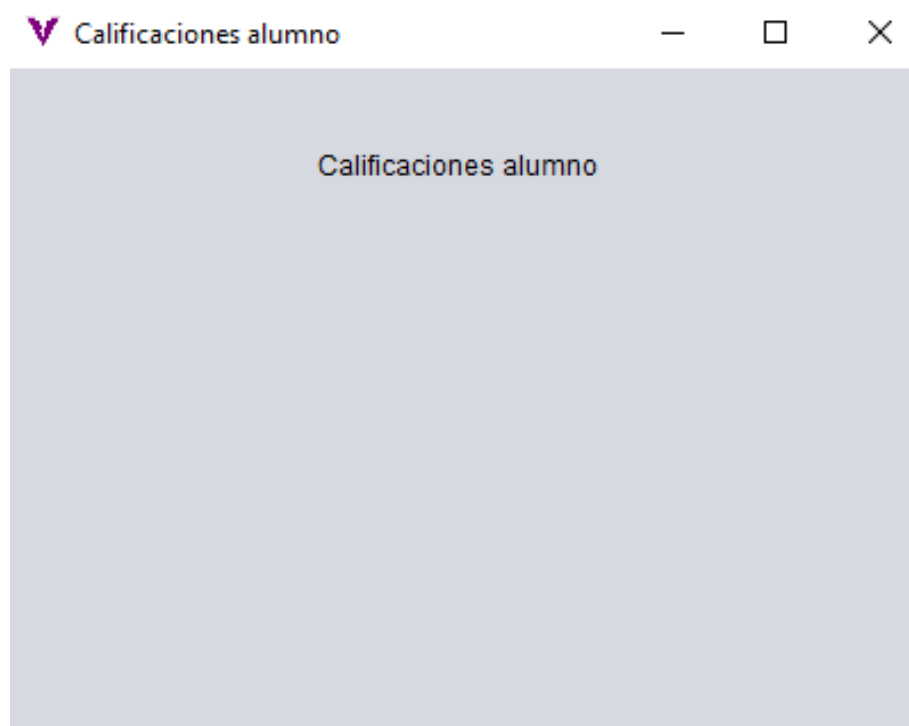


Figura 2.29: JLabel ventana

Luego añadiremos dos JLabel que ponga en text nota1, y nota2, y al lado unos campos de textos, JTextField, y un botón que ponga calcular media, y otro JLabel para mostrar el resultado, como se muestra en la siguiente imagen:

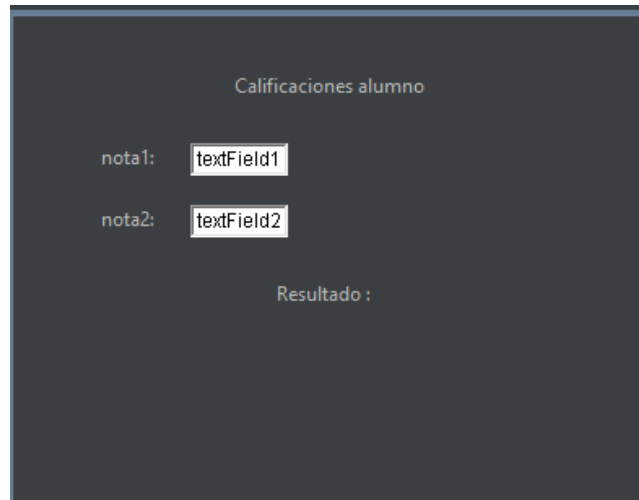


Figura 2.30: Ventana final

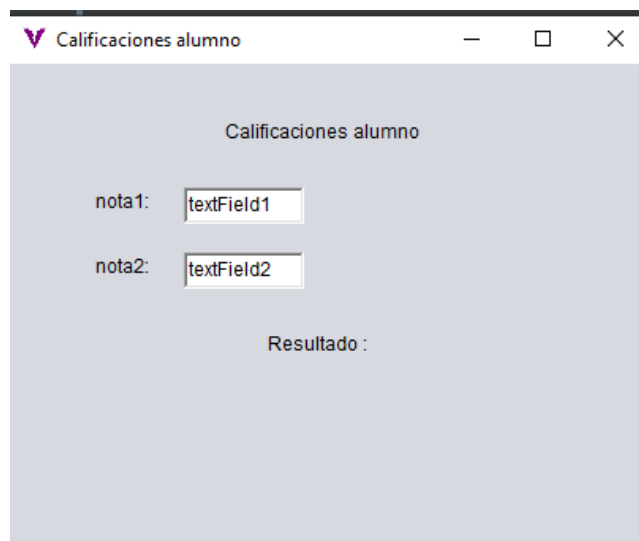


Figura 2.31: Ventana final

Para quitar el texto que viene en Textfield por defecto pinchamos en cada uno de los campos de texto y lo quitamos, además debemos añadir el botón de calcular media:



foreground	■ [0,0,0]
maximumSize	[32767, 32767]
minimumSize	[60, 20]
name	
preferredSize	[60, 20]
text	textField1

Figura 2.32: Texto por defecto

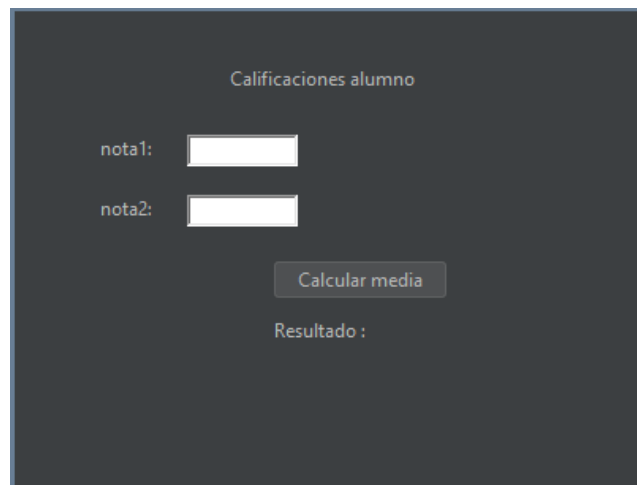


Figura 2.33: Texto por defecto y botón

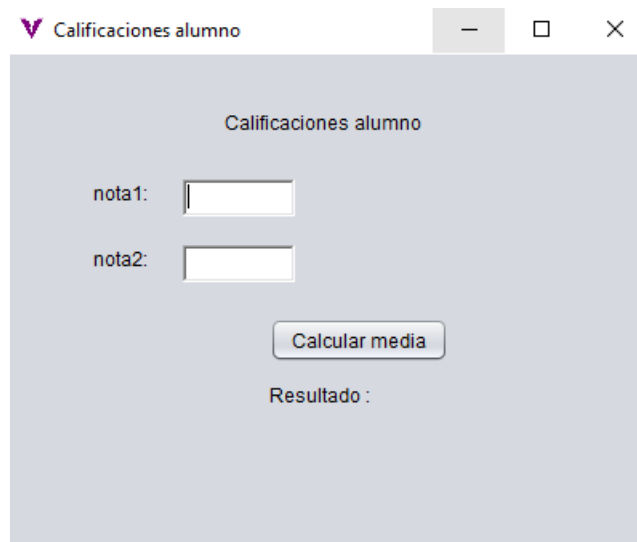


Figura 2.34: Texto por defecto y botón

Ya tenemos la ventana completa, ahora además, lo que nos falta es controlar el evento que cuando se pulse el botón, se calcule la media de los valores que introduzcamos por teclado en las cajas de texto. Antes de nada puntualizar que en las cajas de texto se tendrá siempre datos de tipos String, que luego lo tendremos que pasar a int o float con un método que hay para ello, en principio trabajaremos con números enteros.

Otro detalle a comentar, es que si introducimos caracteres que no son números como letras y otros caracteres que no son números tendremos una excepción, se deja al alumno como tarea complementaria si tiene tiempo realizar un control de errores o tratar las excepciones con bloques try/catch. Pero no nos centraremos ahora en este tema ya que pertenece más bien al ámbito de desarrollo.

Veamos como se trabaja con el evento de clicar el botón, pinchamos el botón en las propiedades, nos vamos entonces a la parte de eventos que hay una pestaña y pulsamos dos veces lo que vemos en la imagen:

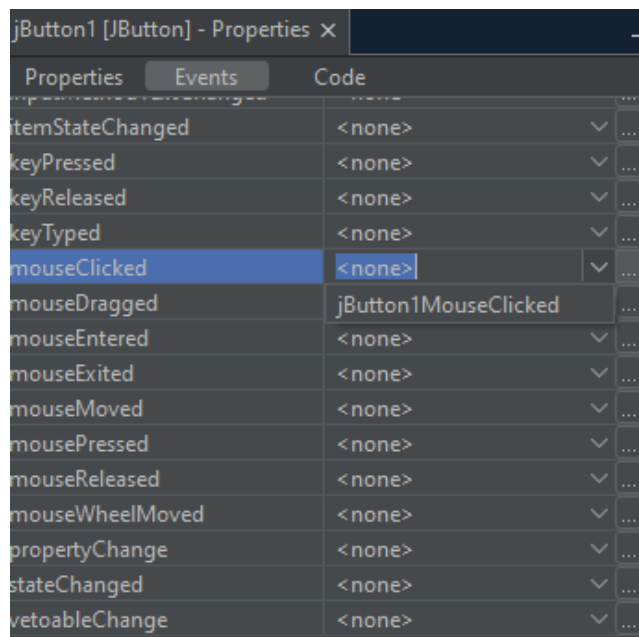


Figura 2.35: Eventos

Pinchando en JButton1MouseClicked, nos lleva directamente al código y añade el método que tratará el evento para implementar, vemos que además incluye un comentario TODO porque está por hacer:

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
}
```

Figura 2.36: Eventos

Por lo que en el código si miramos estará reflejado todo lo que hemos ido colocando en la ventana, los JTextField, con las cajas o campos de texto:

```
private void initComponents() {  
    jLabel1 = new javax.swing.JLabel();  
    jLabel2 = new javax.swing.JLabel();  
    jLabel3 = new javax.swing.JLabel();  
    textField1 = new java.awt.TextField();  
    textField2 = new java.awt.TextField();  
    jLabel4 = new javax.swing.JLabel();  
    jButton1 = new javax.swing.JButton();  
  
    setDefaultCloseOperation(operation: javax.swing.WindowConstants.EXIT_ON_CLOSE);  
    setTitle(title: "Calificaciones alumno");  
    setIconImage(image: new ImageIcon(location: this.getClass().getResource(name:  
  
    jLabel1.setText(text: "Calificaciones alumno");  
    jLabel2.setText(text: "nota1: ");  
    jLabel3.setText(text: "nota2:");  
    jLabel4.setText(text: "Resultado :");  
    jButton1.setText(text: "Calcular media");  
    jButton1.addMouseListener(new java.awt.event.MouseAdapter() {  
        public void mouseClicked(java.awt.event.MouseEvent evt) {  
            jButton1MouseClicked(evt);  
        }  
    });  
}
```

Figura 2.37: Código completo

Para coger los números que pongamos en las cajas de texto, en el esqueleto del evento del botón para poder cogerlos ponemos lo siguiente, se hace con el siguiente método (el método getText para coger el texto de las cajas):

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
  
    String s1, s2;  
  
    s1=textField1.getText();  
    s2=textField2.getText();  
}
```

Figura 2.38: Código evento

Ahora tenemos que pasar el texto o String a número o entero, para ello, hay muchas formas de hacerlo, pero lo vamos a hacer con un método llamado parseInt() que es un método de la clase Integer que se estudió en unidades anteriores. Por lo que el código del evento del botón queda completo como:

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:

    String s1, s2;

    int n1,n2;

    float res;

    s1=textField1.getText();
    s2=textField1.getText();

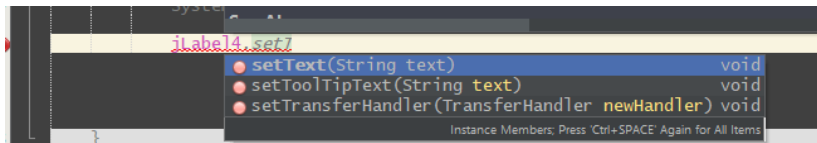
    n1=Integer.parseInt(s1);
    n2=Integer.parseInt(s2);

    res=(n1+n2)/2;

    System.out.println("La media es: " + res);
}
```

Figura 2.39: Código evento completo

Una vez hecho esto, al pulsar el botón se va a ejecutar este código, solo nos queda poner dicho resultado en lugar que salga en el output con el System.out.println lo pondremos en la JLabel de resultado, que claramente será con un setText, un método que pone texto en las Label y en tantas otros componentes de la siguiente forma:



The screenshot shows a code editor with the text `jLabel4.setText` partially entered. A dropdown menu is visible, showing the following options:

- `setText(String text)` void
- `setToolTipText(String text)` void
- `setTransferHandler(TransferHandler newHandler)` void

Below the menu, it says "Instance Members: Press 'Ctrl+SPACE' Again for All Items".

Figura 2.40: Resultado

```
private void jButton1MouseClicked(java.awt.event.MouseEvent
// TODO add your handling code here:

String s1, s2;

int n1,n2;

float res;

s1=textField1.getText();
s2=textField1.getText();

n1=Integer.parseInt(s1);
n2=Integer.parseInt(s2);

res=(n1+n2)/2;

System.out.println("La media es: " + res);

jLabel4.setText("La media es: " + res);
}
```

Figura 2.41: Código final

Vamos a probar el código para ver el resultado:

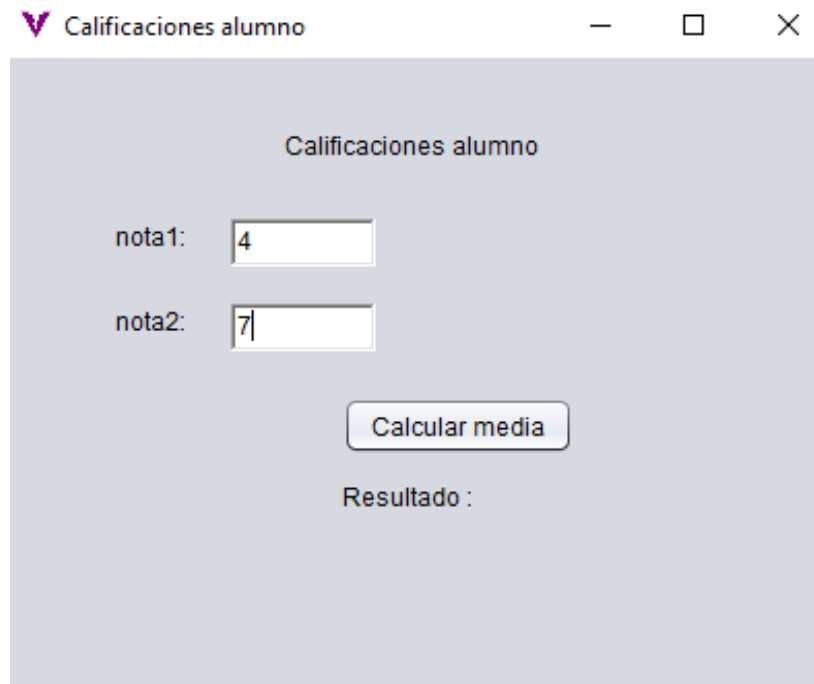


Figura 2.42: Programa final.

Cuidado porque hay una errata en `s2=textFiedl2` no `textField1`, hay que corregirlas en las imágenes. Porque sino en los dos casos te toma el texto del la primera cajita. El código hay que pasar los números enteros a reales o float con un casting de tipos poniendo delante (`float`) como se muestra en la imagen:

```
res=((float)n1+(float)n2)/2;
```

Figura 2.43: Números reales.

El resultado final es el siguiente:

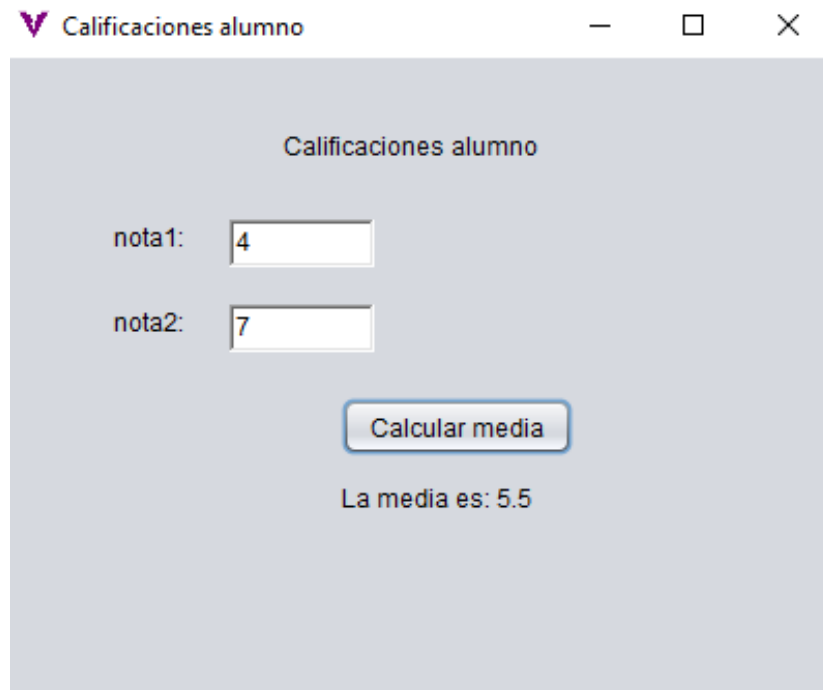


Figura 2.44: Resultado final.

Se le propone al alumno modificar el programa para que en las cajas podamos escribir números con decimales y que haga la media. Una pista es que se puede usar `Float.parseFloat(String)`. Cambiarlo y mostrar una captura de pantalla.

## 3.2. Prácticas Interfaces propuestas

En el apartado anterior, se ha realizado una práctica de un ejercicio guiado, en la siguiente se le va a proponer al alumno realizar dos de ellos y que realice capturas de pantallas comentadas brevemente:

**Problema 1:** Se trata de realizar un ejercicio parecido al anterior con números float, con 5 resultados de aprendizajes (5 Labels con RA1:, RA2:, etc.. ), 5 campos de texto al lado de cada label y a la derecha de cada campo de texto otros 5 campos de label con el texto: ( %15, %23, etc), indicando el porcentaje de cada RA. En un Label se pondrá el resultado de la media ponderada al pulsar un botón y en otro ACTO o NO ACTO.

**Problema 2:** Adivinar un número entero entre 1 y 100 aleatorio y el usuario irá introduciendo valores y en un Label aparecerá es mayor, es menor o has acertado y en otro label el número de intentos o cuántos intentos quedan, se deja a voluntad del

alumno. Para crear un número aleatorio entre 1 y 100 en Java es con: `numero = (int) (Math.random() * 100) + 1`.

## 4. Interfaces en Java

### 4.1. ¿Qué son los interfaces en Java?

Una interfaz en Java es un conjunto de declaraciones de métodos que luego pueden ser implementadas por las clases de Java mediante la cláusula `implements` similar a `extends` para las herencias.

Las interfaces son muy útiles cuando se desea definir métodos que van a hacer usados en varias clases, por ejemplo, si usamos un método llamado `dibujar()`, en una clase puede que lo use para dibujar un rectángulo, en otra para dibujar un círculo y en otra un triángulo, con lo que en cada una de ellas se va a implementar un método distinto.

Veamos un ejemplo en Java usando Netbeans:

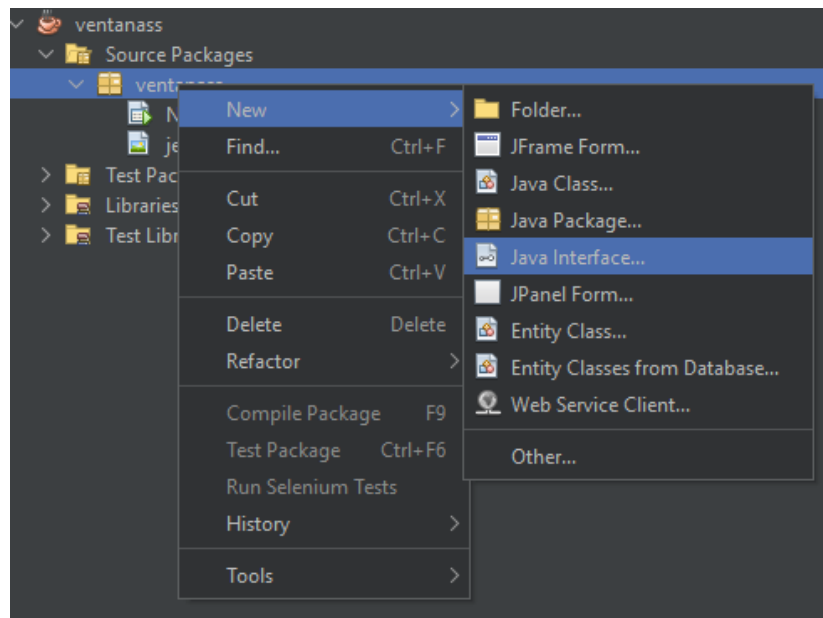


Figura 4.1: Netbeans crear interface

Luego le damos un nombre a la interface como se muestra en este menú:



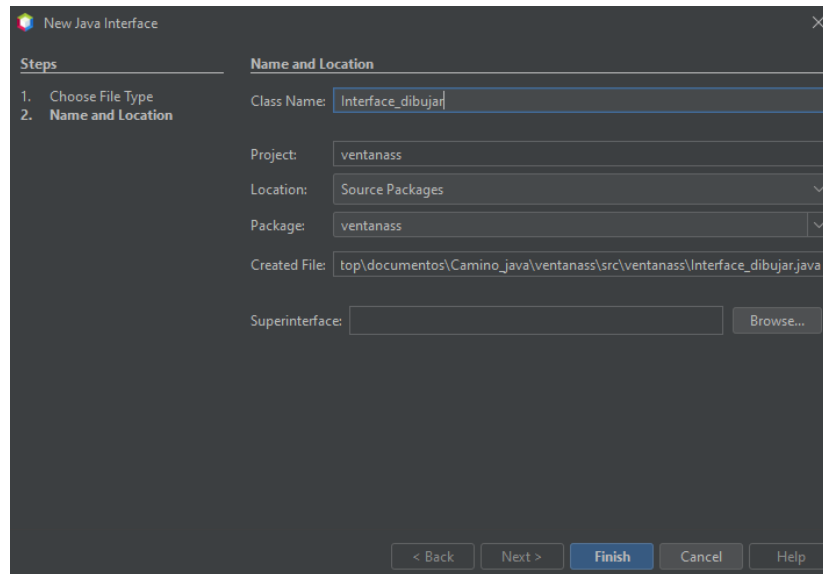


Figura 4.2: Netabeans crear interface

Nos aparecerá algo como este código:

```
package ventanass;

/**
 *
 * @author Abel
 */
public interface Interface_dibujar {

}
}
```

Figura 4.3: Netabeans crear interface

Por lo que se va a mostrar a continuación como se crea la interfaz, puede tener métodos y atributos o variables, pero lo normal es métodos, no es lógico incluir atributos que van a usar clases pero se puede.

El código queda de la siguiente forma:

```
2      package ventanass;
3
4      /**
5       *
6       * @author Abel
7       */
8      public interface Interface_dibujar {
9
10         public void dibujar();
11
12         public int getNumero();
13
14         public void sumar(int a, int b);
15
16     }
```

Para implementar esta interfaz en una clase, se crea una clase cualquiera en java con new, nueva clase y se puede ver el código como sigue:

```
17     package ventanass;
18
19     /**
20      *
21      * @author Abel
22      */
23     public class clasee implements Interface_dibujar{
24
25         public void dibujar()
26         {}
27
28         public int getNumero()
29         {
30
31             return 0;}
32
33         public void sumar(int a, int b)
34         {}
35
36     }
```

Para implementar varios interfaces, por ejemplo 2 interfaces en una clase se ponen separados por coma:

```
37     public class clasee implements interfaz1, interfaz2{
38
39     }
```

En la sección siguiente se propondrá algún ejercicio sobre interfaces en Java.

## 4.2. Ejercicios propuestos interfaces

Cuestiones y ejercicios propuestos para responder:

**Cuestión1:** ¿Es posible incluir método privados en una interfaz? Razone la respuesta.

**Cuestión2:** ¿Qué es una clase abstracta y cómo se puede usar en las interfaces?

**Ejercicio1:** Implemente una interfaz llamada `calcular_nombrealumno` que tenga varios métodos: `operacion1`, `operacion2`, `mostrar_resultado`, e implementar una clase de ejemplo que los implemente.

**Ejercicio2:** En la siguiente imagen se usa la cláusula `implements` en las clases swing que se vieron anteriormente, ¿qué significan?:

```
41     public class clasee implements ActionListener{
42
43         @Override
44         public void actionPerformed(ActionEvent e) {
45             throw new UnsupportedOperationException("Not
                supported yet."); // Generated from nbfs://
                nbhost/SystemFileSystem/Templates/Classes/
                Code/GeneratedMethodBody
46         }
```

## 5. Entrega

La entrega de la práctica se enviará por la plataforma moodle con el formato siguiente: nombrealumno\_apellido1\_apellido2.pdf, deberá incluir todas las capturas comentadas brevemente lo que se ha hecho en ellas con una o dos líneas. El plazo queda fijado un día antes de las evaluaciones.

Además se tendrá en cuenta la presentación del documento de la práctica, y aportaciones extras que se hagan.

## 6. Bibliografía

- [1] web oracle <https://docs.oracle.com/javase/8/docs/api///?javax/swing/package-summary.html>
- [2] La guía definitiva de Java Swing 3ª edición 2005 autor: JOHN ZUKOWSKI  
<https://www.cs.buap.mx/~ygalicia/Libros/JavaSwing.pdf>
- [3] O'Reilly segunda edición 2002 Java Swing [https://nuleren.be/edocumenten/OReilly\\_Java\\_Swing.pdf](https://nuleren.be/edocumenten/OReilly_Java_Swing.pdf)
- [4] Swing Manning Swing Una guía rápida con ejemplos autores: Matthew Robinson, Pavel Yorobiev [https://box.cs.istu.ru/public/docs/other/\\_Unsorted/new/Java/Java%20Swing%20\(Manning\).pdf](https://box.cs.istu.ru/public/docs/other/_Unsorted/new/Java/Java%20Swing%20(Manning).pdf)
- [5] GUIs en Java Autor: Antonio la Torre UPM, Universidad Politécnica de Madrid [https://laurel.datsi.fi.upm.es/\\_media/docencia/cursos/java/2012/guis\\_en\\_java-lpp\\_2012\\_.pdf](https://laurel.datsi.fi.upm.es/_media/docencia/cursos/java/2012/guis_en_java-lpp_2012_.pdf)