

TEMA 2. INSTALACIÓN Y USO DE ENTORNOS

2.4. EDITORES EN MODO CONSOLA Y USO DE LIBRERÍAS

ENTORNOS DE DESARROLLO

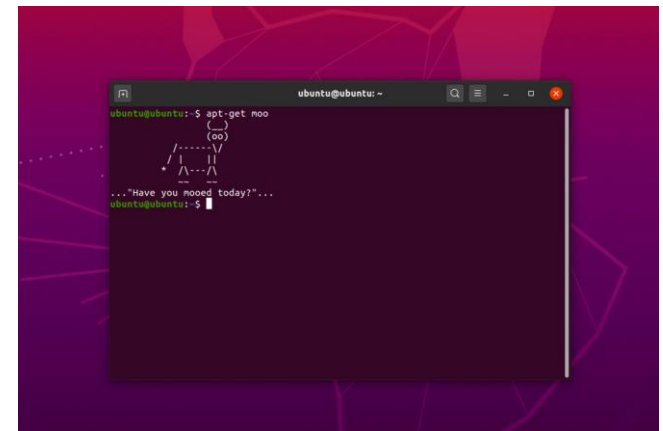
```
Macintosh SSD - top - 80x24
Processes: 430 total, 22 running, 408 sleeping, 1300 threads    10:55:00
Load Avg: 7.88, 15.32, 21.50 CPU usage: 51.50% user, 26.60% sys, 21.80% idle
SharedLibs: 349M resident, 104M data, 61M linkedit.
MemRegions: 158911 total, 2300M resident, 285M private, 1453M shared.
PhysMem: 8395M used (1622M wired), 52M unused.
VM: 10T vsize, 2320M framework vsize, 0(0) swapins, 0(0) swapouts.
Networks: packets: 113962/117M in, 35531/4878K out.
Disks: 386785/7137M read, 65994/1135M written.

PID  COMMAND   %CPU  TIME    #TH  #PQ  #PORT  MEM    PAGE  CMPS  PGPF  PPID
151  WindowServer 36.3  04:33.72 10/1  5    1743  153M-  44K+  5716K 151  1
0    kernel_task 11.6  04:38.74 165/4  0     0    0B-   0B   0B   0  0
1744  top         0.0   00:01.95 1/1    0    20+   3540K+ 0B   0B   1744 1738
1746  screencaptr 4.5   00:00.35 3      2    52+   2740K+ 0B   0B   327 327
193   top         3.2   00:04.62 4      3    54+   2732K+ 28K   8192B 193 1
335   ANVisualSupp 3.1   00:17.66 7      2    231   10M    16K    32K   335 1
194   trustd     2.3   00:12.62 3      2    109   3556K+ 466K   212K  194 1
1590  Music      2.3   00:22.54 16     3    497   88M    6856K  48M   1590 1
450   Macs Fan Cn 1.0   00:20.07 7      2    230   12M+   16K-   104K  450 1
1     launchd    1.5   00:12.28 3      2    4617+ 22M+   0B    2724K 1  0
120   coreaudiod 1.4   00:10.03 7      1    407   8224K  0B    40K   120 1
163   loginwindow 1.4   00:04.50 3      2    356   48M-   848K   15M   163 1
1715  Terminal   1.4   00:04.20 9      4    270   45M    292K  0B    1715 1
122   launchserv 1.3   00:02.98 6      5    550+  2820K+ 0B    8192B 122 1
```

EDITORES EN MODO CONSOLA

Editores en modo consola

- Los **editores por consola** son aplicaciones que nos permiten crear código desde la terminal.
- Para esto usaremos una terminal Linux.
- Existen terminales online como la siguiente:
<https://bellard.org/jslinux/vm.html?url=alpine-x86.cfg&mem=192>



Editores en modo consola

- Las aplicaciones que usaremos serán **nano** y **vi**.
- Comprobar si los tenemos instalados:

```
nano --version
```

- En caso de que no estén instalados:

```
sudo apt-get install nano
```

```
sudo apt-get install vi
```

Usar nano

- Para crear un fichero usamos la orden:

`nano nombre_fichero.extension`

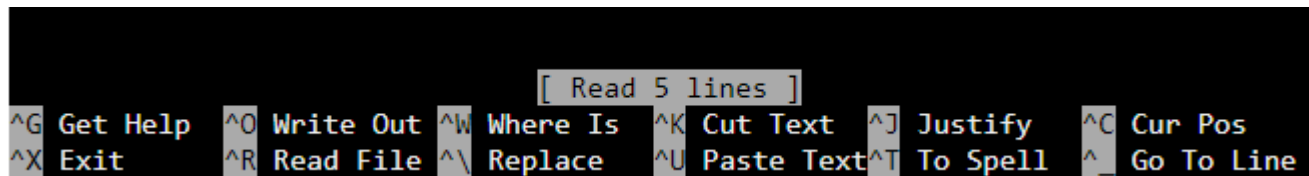
ej: `nano HolaMundo.java`

```
localhost:~# nano HolaMundo.java
```

```
GNU nano 4.9.3      HolaMundo.java      Modified
public class HolaMundo{
    public static void main(String[] args){
        System.out.println("Hola");
    }
}
```

Usar nano

- Para guardar usamos: CTRL+O
- Para salir usamos: CTRL+X
- El resto de comandos se pueden ver en la parte inferior:



A screenshot of the nano text editor's command menu. The menu is displayed in a black box with white text. At the top, there is a prompt "[Read 5 lines]". Below it, the commands are arranged in two rows. The first row contains: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, and ^C Cur Pos. The second row contains: ^X Exit, ^R Read File, ^\ Replace, ^U Paste Text, ^T To Spell, and ^ Go To Line.

```
[ Read 5 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text ^T To Spell  ^ Go To Line
```

Usar vi

- Para crear un fichero usamos la orden:

`vi nombre_fichero.extension`

ej: `vi hola.java`

```
localhost:~# vi hola.java
```

Usar vi

- Los comandos en vi no aparecen en la parte inferior:

i: comenzar a escribir texto

```
public class Hola{  
    public static void main (String[] args){  
        System.out.println("Hola");  
    }  
}  
~
```


Usar vi

- Los comandos en vi no aparecen en la parte inferior:

Ahora pulsamos **Esc** para salir del modo edición.

:wq para guardar y salir.

:w para guardar y no salir.

:q! salir sin guardar.

Podéis ver el resto de los comandos aquí:

<https://www.redhat.com/sysadmin/introduction-vi-editor>

Usar vi

- Ahora podemos compilar y ejecutar el fichero java:

```
> javac hola.java  
> java hola
```

¡OJO!: en la terminal web no funciona.

Entorno VSC.

Uso de librerías externas.



VSC. Librerías

- Las librerías en JAVA tienen el formato .jar
- Nos permiten obtener funcionalidad ya creada por otros programadores.

6 LIBRERÍAS PARA JAVA

Una librería (o biblioteca) es un **conjunto de funciones** creada para resolver una necesidad específica.

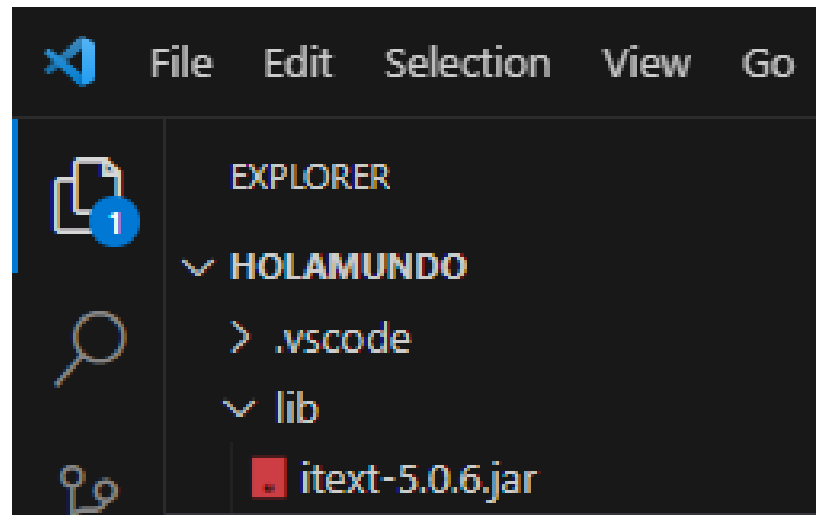
- 1 GOOGLE GUAVA**
Mejora del flujo de trabajo destinado a tareas comunes. (E/S, caché, primitivas, entre otros).
- 2 RXJAVA**
Crea aplicaciones para **reaccionar** a los flujos de datos en tiempo real.
- 3 MPANDROIDCHART**
Crea gráficos de líneas, barras, radares y burbujas para Android.
- 4 FASTJSON**
Convierte objetos Java en su representación JSON y viceversa.
- 5 MOCKITO**
Librería de código abierto para **simular pruebas unitarias**.
- 6 JUNIT**
Esta es la librería más **usada para pruebas**.

Aprende Java desde cero hasta nivel Jedi en:
ed.team/cursos/java

EDteam

VSC. Librería itext

- En Aules tenéis el fichero *itext-5.0.6.jar*.
- Esta librería nos permite crear documentos PDF desde el IDE con JAVA.
- Para usar librerías las pegamos dentro de la carpeta `lib` del proyecto:



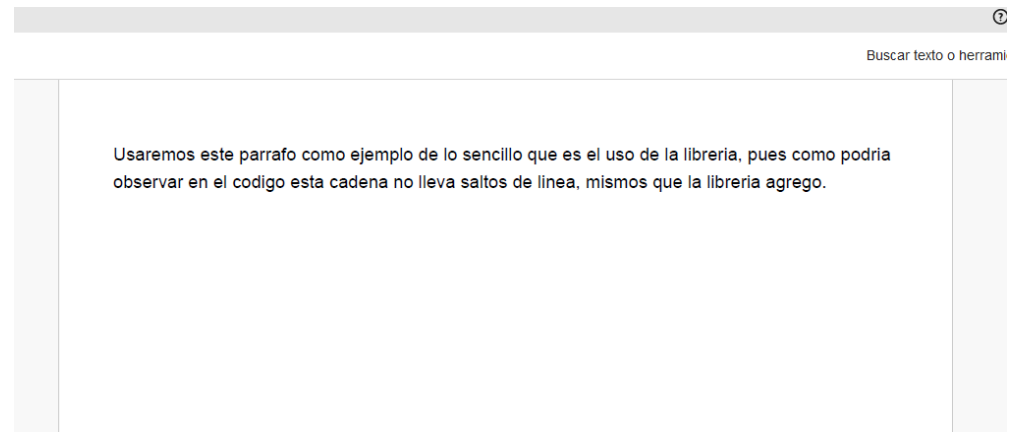
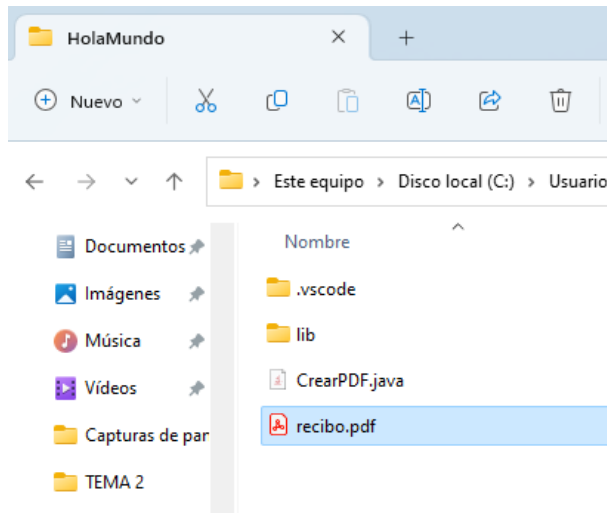
VSC. Código librería itext

```
File Edit Selection View Go Run Terminal Help
HOLAMUNDO
  .vscode
  lib
    itext-5.0.6.jar
    CrearPDF.java

CrearPDF.java
1  import com.itextpdf.text.Document;
2  import com.itextpdf.text.pdf.PdfWriter;
3  import com.itextpdf.text.Paragraph;
4  import java.io.FileOutputStream;
5
6  public class CrearPDF {
7      public void añadirTexto() throws Exception {
8          Document document = new Document();
9          String parrafo = "Usaremos este parrafo como ejemplo de lo sencillo que es el uso de la libreria, "
10             + "pues como podria observar en el codigo esta cadena no lleva saltos de linea, "
11             + "mismos que la libreria agrego.";
12          PdfWriter.getInstance(document, new FileOutputStream(name:"recibo.pdf"));
13          document.open();
14          document.add(new Paragraph(parrafo));
15          document.close();
16      }
17
18      public static void main(String args[]) {
19          try {
20              CrearPDF p = new CrearPDF();
21              p.añadirTexto();
22          } catch (Exception e) {
23              System.out.println(e);
24          }
25      }
26  }
```

VSC. Generar PDF

- Cuando ejecutamos el código aparece el PDF en la carpeta del proyecto:



VSC. Visualizar PDF

- Si intentamos visualizar el PDF desde el editor no nos será posible:

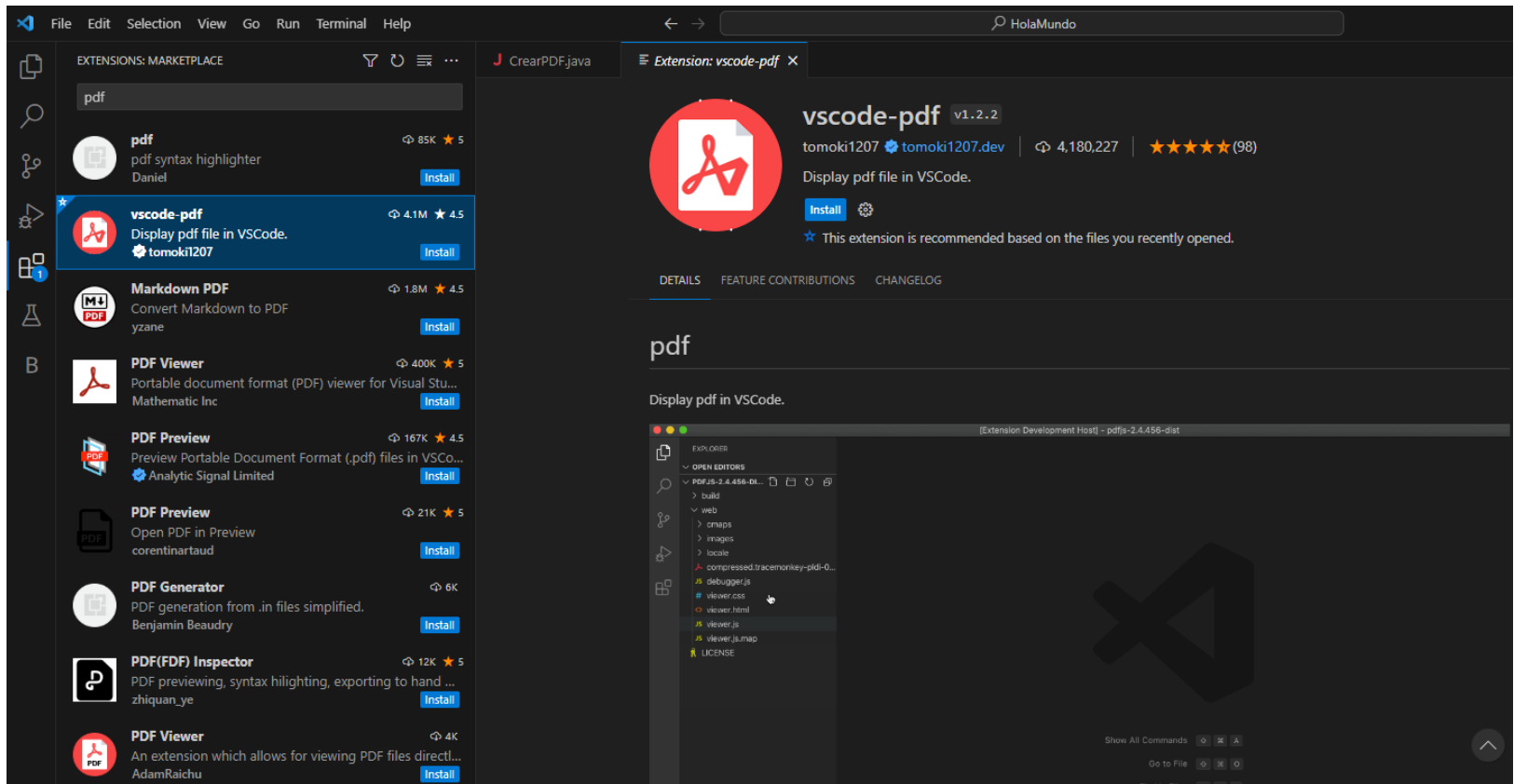
VSC. Visualizar PDF

- Si intentamos visualizar el PDF desde el editor no nos será posible:

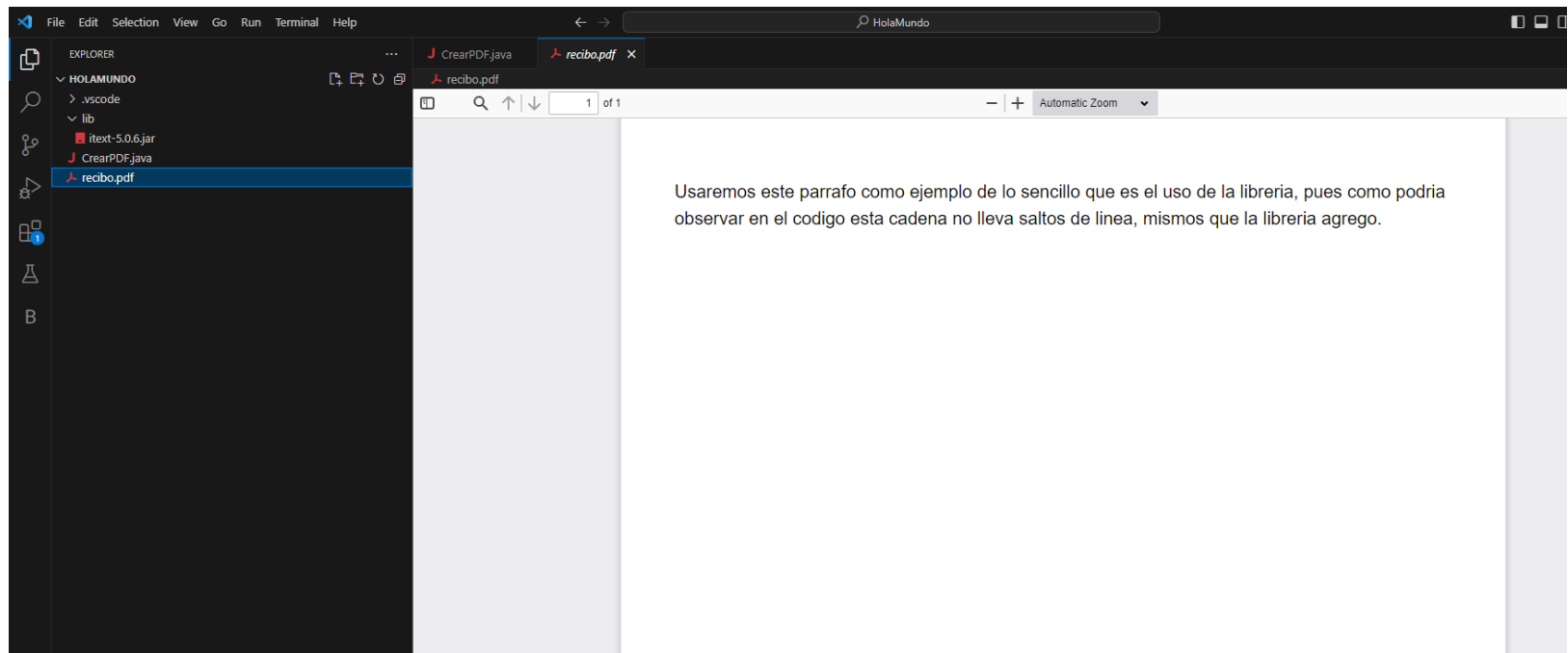
[illegible]

VSC. Visualizar PDF

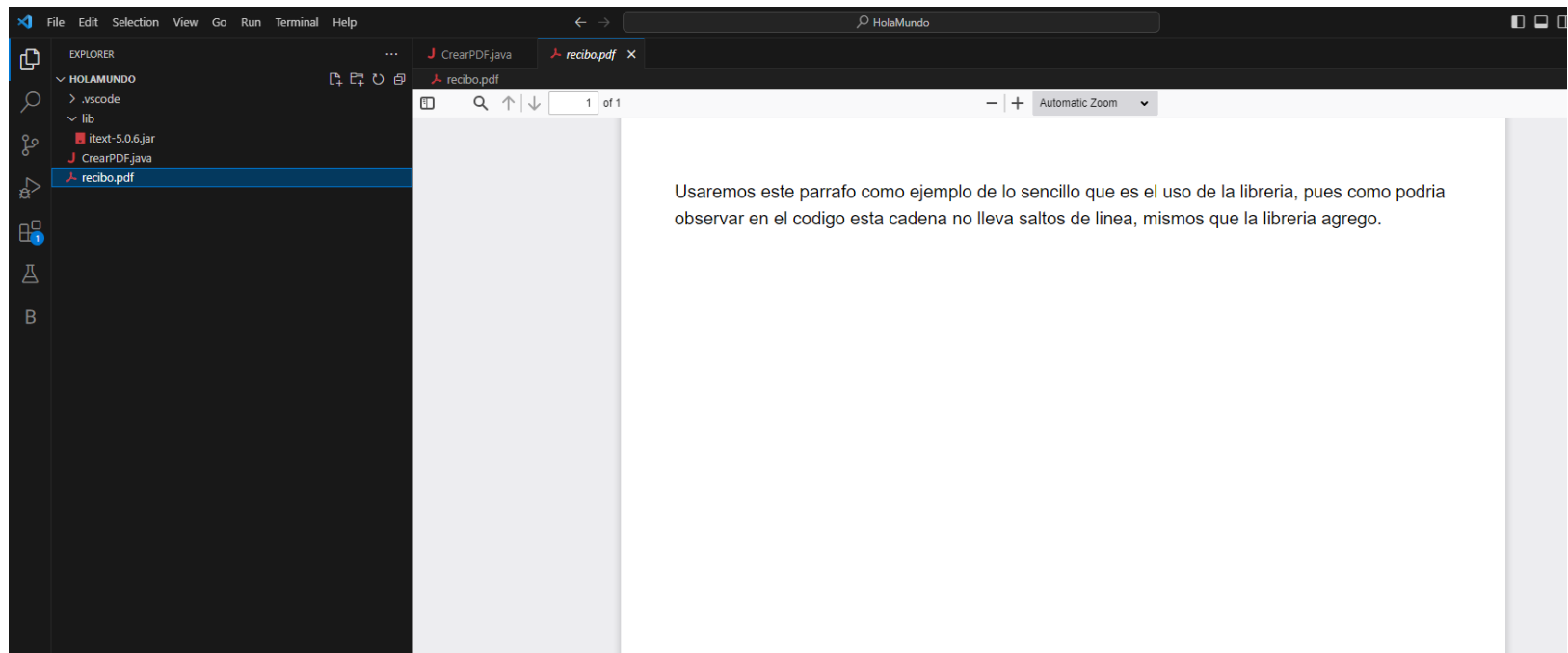
- Podemos instalar esta extensión para visualizar ficheros:



VSC. Visualizar PDF



VSC. Visualizar PDF



Entorno VSC.

Creación de interfaces gráficas sencillas.



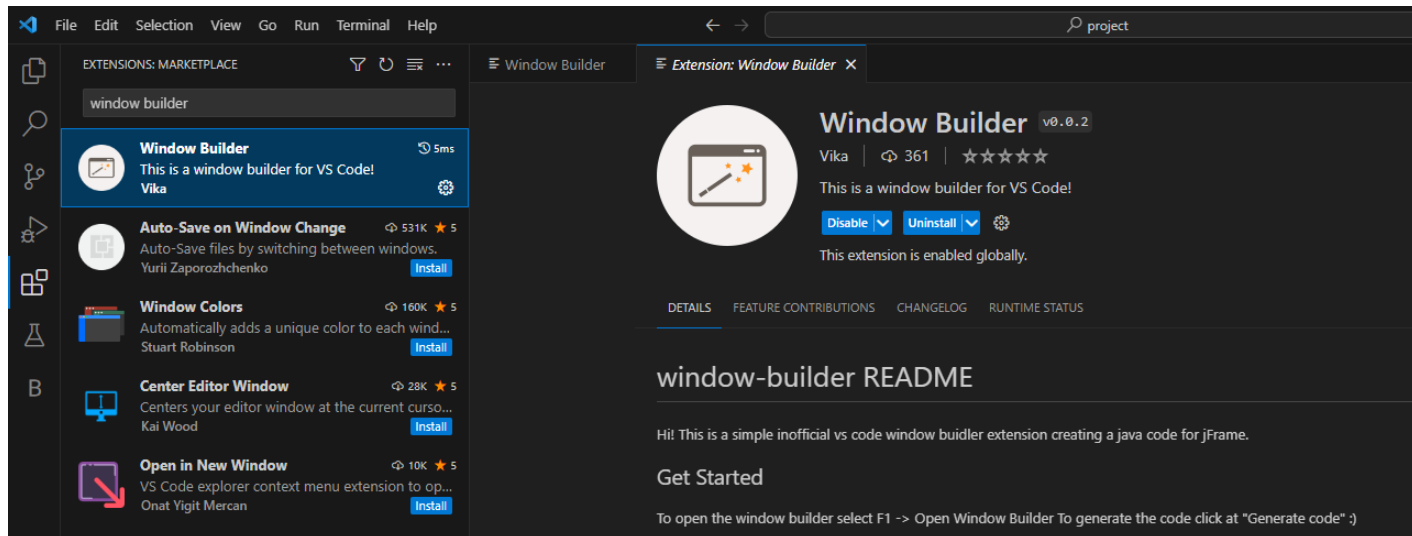
GUI en VSC

- VSC nos permite crear interfaces gráficas (Graphical User Interface) mediante diversas herramientas:
 - JAVA FX
 - WindowBuilder



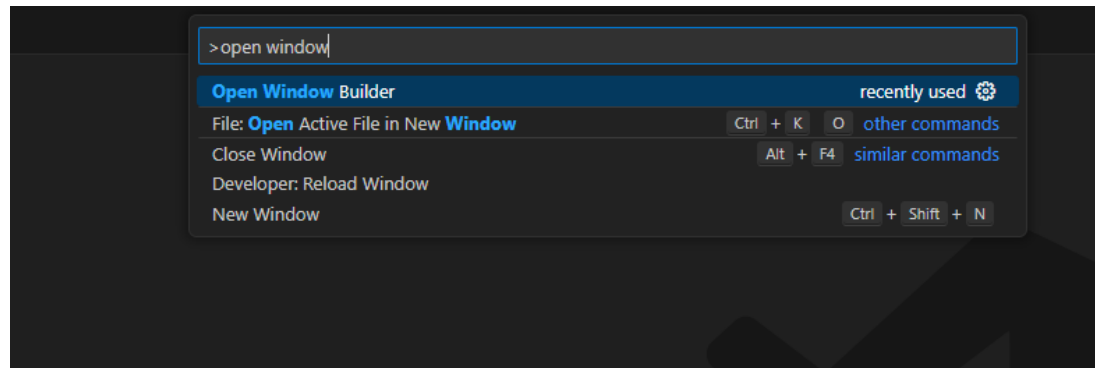
GUI en VSC

- Window Builder es un plugin para Eclipse pero que se ha portado a VSC.
- Para usarlo lo podemos buscar en las extensiones:



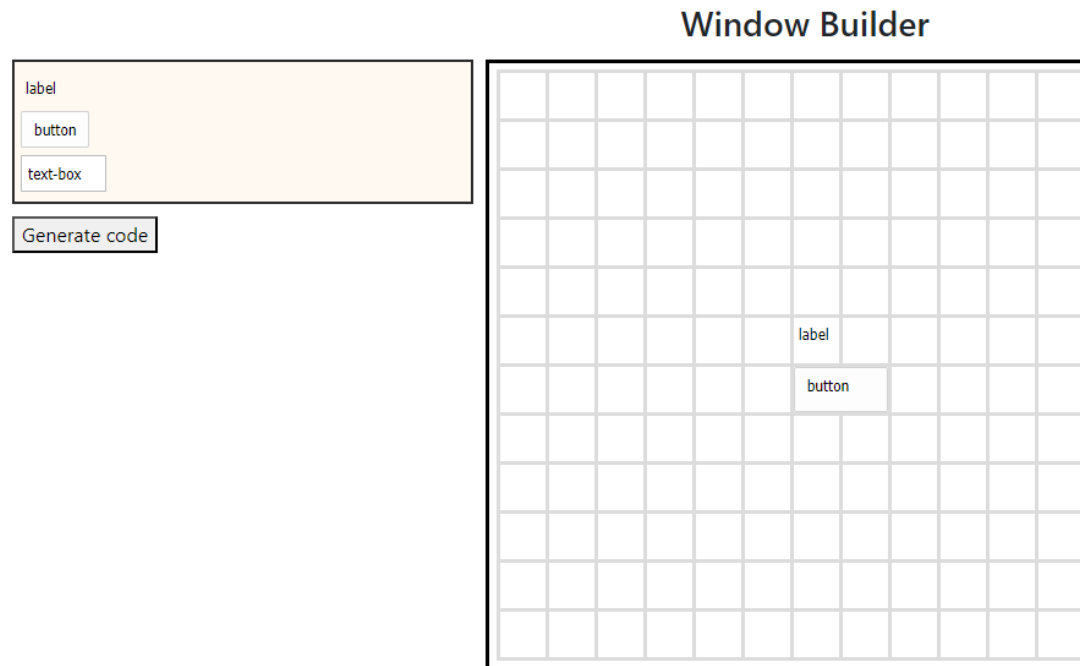
GUI en VSC

- Creamos un proyecto Java normal y pulsamos F1.
- Buscamos Open Window Builder.



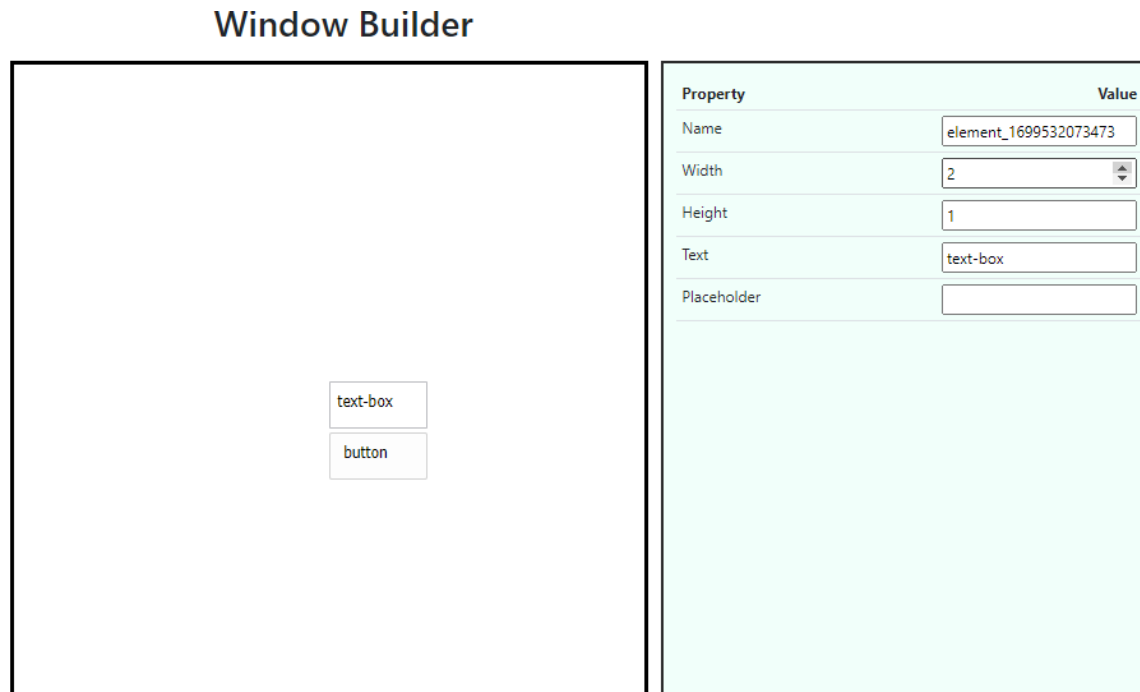
GUI en VSC

- Podemos crear interfaces moviendo los componentes de la izquierda al grid.



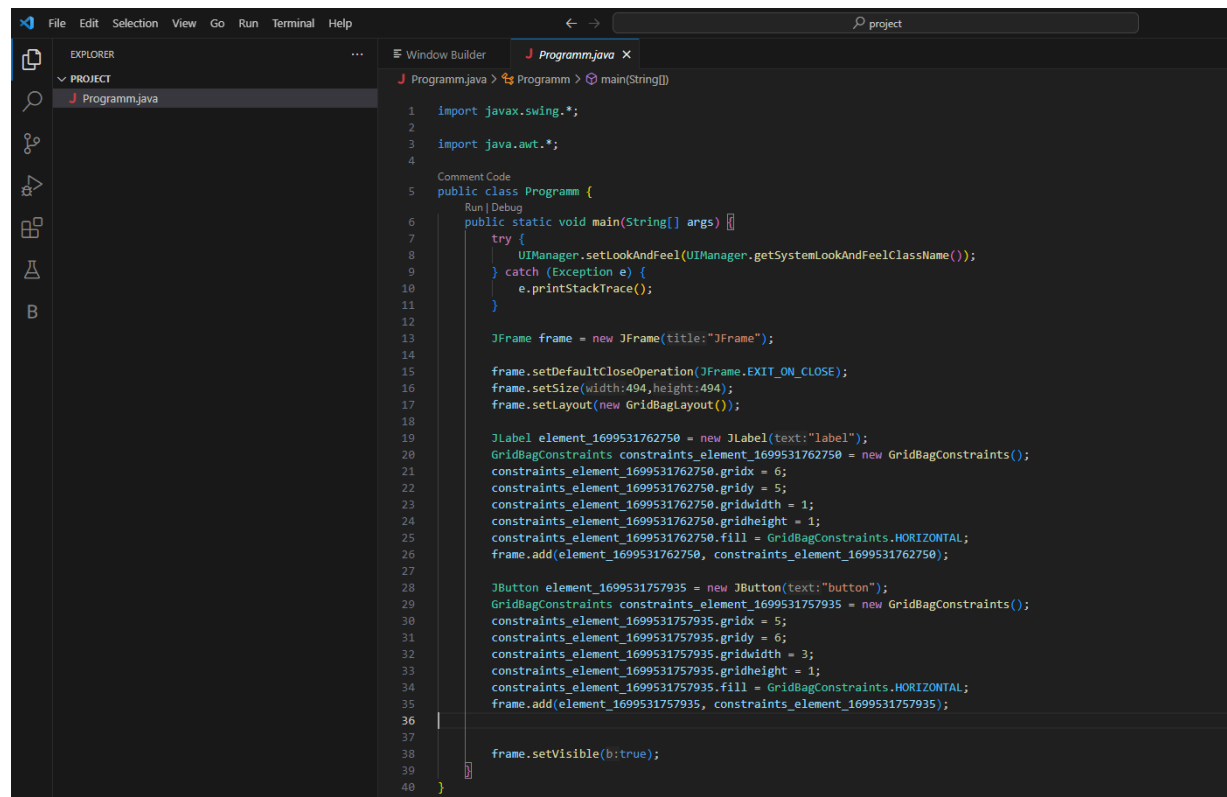
GUI en VSC

- Podemos redimensionar los componentes mediante las opciones del menú derecho.



GUI en VSC

- Se genera un código como el siguiente:

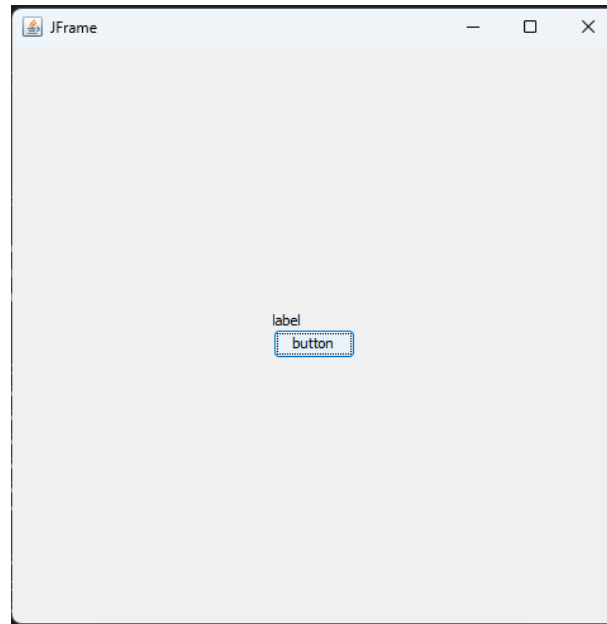


The screenshot shows the Visual Studio Code editor with a Java file named `Programm.java` open. The Explorer sidebar on the left shows the project structure with `Programm.java` selected. The editor window displays the following Java code:

```
1  import javax.swing.*;
2
3  import java.awt.*;
4
5  Comment Code
6  public class Programm {
7      Run | Debug
8      public static void main(String[] args) {
9          try {
10             UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
11         } catch (Exception e) {
12             e.printStackTrace();
13         }
14
15         JFrame frame = new JFrame(title:"JFrame");
16
17         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18         frame.setSize(width:494,height:494);
19         frame.setLayout(new GridBagLayout());
20
21         JLabel element_1699531762750 = new JLabel(text:"label");
22         GridBagConstraints constraints_element_1699531762750 = new GridBagConstraints();
23         constraints_element_1699531762750.gridx = 6;
24         constraints_element_1699531762750.gridy = 5;
25         constraints_element_1699531762750.gridwidth = 1;
26         constraints_element_1699531762750.gridheight = 1;
27         constraints_element_1699531762750.fill = GridBagConstraints.HORIZONTAL;
28         frame.add(element_1699531762750, constraints_element_1699531762750);
29
30         JButton element_1699531757935 = new JButton(text:"button");
31         GridBagConstraints constraints_element_1699531757935 = new GridBagConstraints();
32         constraints_element_1699531757935.gridx = 5;
33         constraints_element_1699531757935.gridy = 6;
34         constraints_element_1699531757935.gridwidth = 3;
35         constraints_element_1699531757935.gridheight = 1;
36         constraints_element_1699531757935.fill = GridBagConstraints.HORIZONTAL;
37         frame.add(element_1699531757935, constraints_element_1699531757935);
38
39         frame.setVisible(true);
40     }
41 }
```

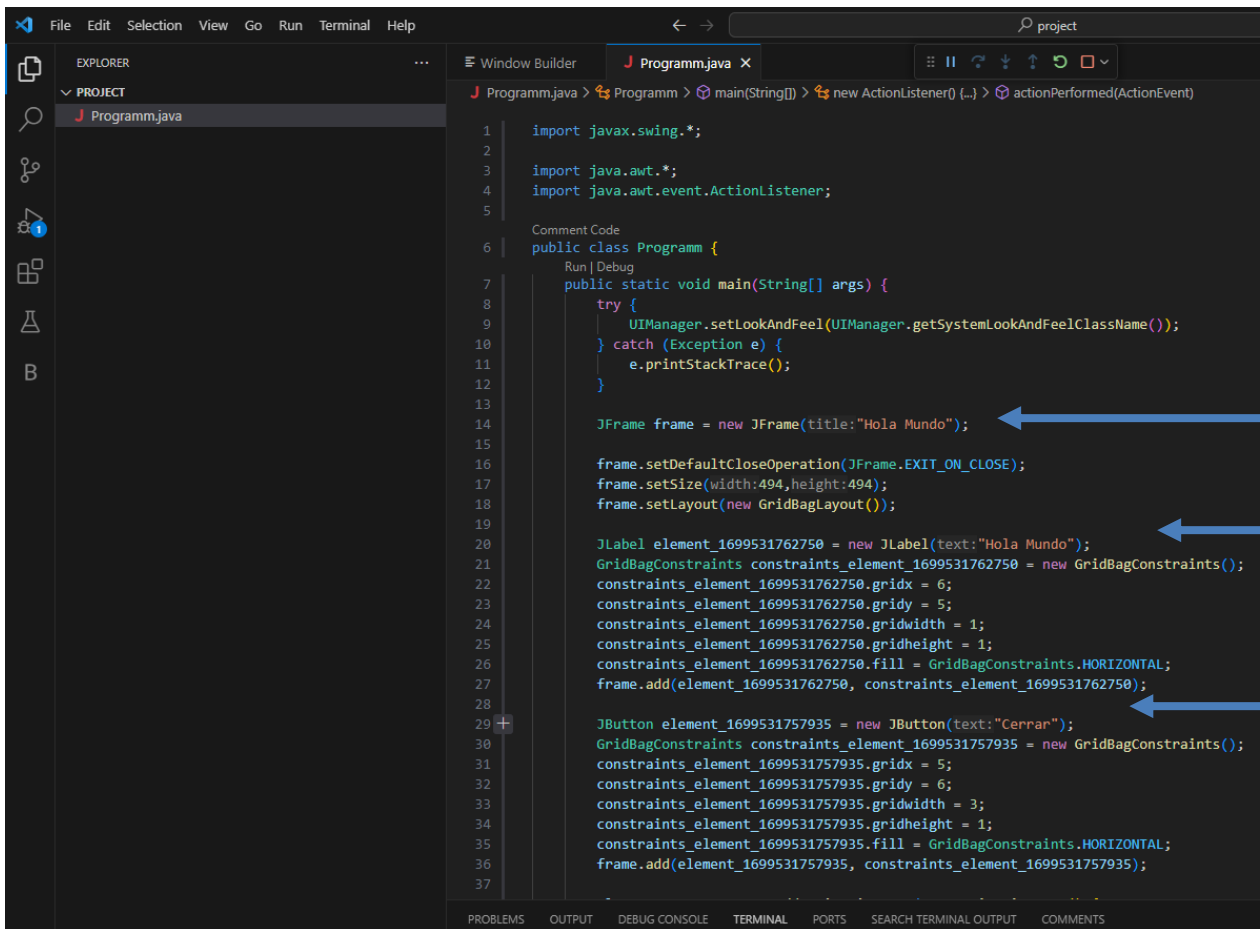
GUI en VSC

- Si lo ejecutamos podemos ver la interfaz:

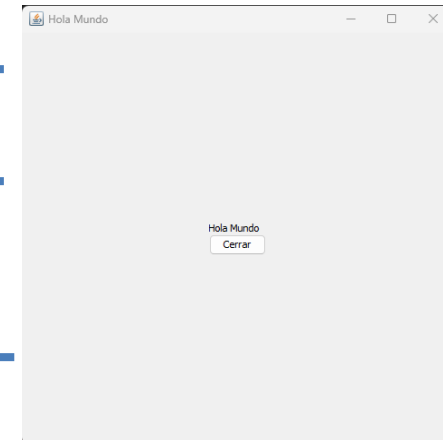


GUI en VSC

- Podemos cambiar el código para mostrar otra información:



```
1  import javax.swing.*;
2
3  import java.awt.*;
4  import java.awt.event.ActionListener;
5
6  Comment Code
7  public class Programm {
8      Run | Debug
9      public static void main(String[] args) {
10         try {
11             UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
12         } catch (Exception e) {
13             e.printStackTrace();
14         }
15
16         JFrame frame = new JFrame(title:"Hola Mundo");
17
18         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19         frame.setSize(width:494,height:494);
20         frame.setLayout(new GridBagLayout());
21
22         JLabel element_1699531762750 = new JLabel(text:"Hola Mundo");
23         GridBagConstraints constraints_element_1699531762750 = new GridBagConstraints();
24         constraints_element_1699531762750.gridx = 6;
25         constraints_element_1699531762750.gridy = 5;
26         constraints_element_1699531762750.gridwidth = 1;
27         constraints_element_1699531762750.gridheight = 1;
28         constraints_element_1699531762750.fill = GridBagConstraints.HORIZONTAL;
29         frame.add(element_1699531762750, constraints_element_1699531762750);
30
31         JButton element_1699531757935 = new JButton(text:"Cerrar");
32         GridBagConstraints constraints_element_1699531757935 = new GridBagConstraints();
33         constraints_element_1699531757935.gridx = 5;
34         constraints_element_1699531757935.gridy = 6;
35         constraints_element_1699531757935.gridwidth = 3;
36         constraints_element_1699531757935.gridheight = 1;
37         constraints_element_1699531757935.fill = GridBagConstraints.HORIZONTAL;
38         frame.add(element_1699531757935, constraints_element_1699531757935);
39     }
40 }
```



GUI en VSC

- Podemos hacer que el botón de cerrar funcione añadiendo código. En este caso se hace mediante un **LISTENER**.

```
constraints_element_1699531762750.gridheight = 1;
constraints_element_1699531762750.fill = GridBagConstraints.HORIZONTAL;
frame.add(element_1699531762750, constraints_element_1699531762750);

JButton element_1699531757935 = new JButton(text:"Cerrar");
GridBagConstraints constraints_element_1699531757935 = new GridBagConstraints();
constraints_element_1699531757935.gridx = 5;
constraints_element_1699531757935.gridy = 6;
constraints_element_1699531757935.gridwidth = 3;
constraints_element_1699531757935.gridheight = 1;
constraints_element_1699531757935.fill = GridBagConstraints.HORIZONTAL;
frame.add(element_1699531757935, constraints_element_1699531757935);

element_1699531757935.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        frame.setVisible(b:false);
    }
});

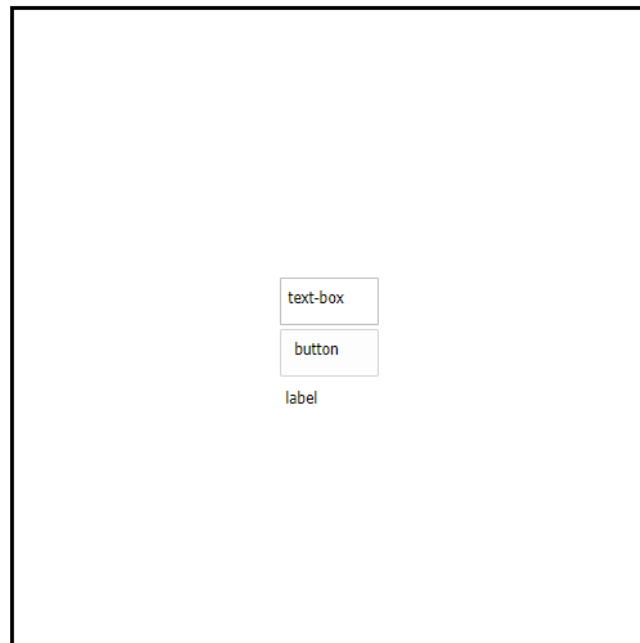
frame.setVisible(b:true);
}
```

GUI en VSC

- **OTRO EJEMPLO:**

Vamos a preguntar el nombre de forma gráfica y al pulsar aceptar
Mostraremos: ¡Hola, NOMBRE!

Window Builder



GUI en VSC

- OTRO EJEMPLO:

Vamos a preguntar el nombre de forma gráfica y al pulsar aceptar
Mostraremos: ¡Hola, NOMBRE!

```
constraints_element_1699533817409.gridheight = 1;  
constraints_element_1699533817409.fill = GridBagConstraints.HORIZONTAL;  
frame.add(element_1699533817409, constraints_element_1699533817409);  
  
element_1699533808695.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        String nombre = element_1699533811655.getText();  
        element_1699533817409.setText("¡Hola, " + nombre + "!");  
    }  
});  
  
frame.setVisible(b:true);  
}
```


GUI en VSC

- **OTRO EJEMPLO:**

Vamos a preguntar el nombre de forma gráfica y al pulsar aceptar
Mostraremos: ¡Hola, NOMBRE!

