

PROGRAMACIÓN

UP8. PROGRAMACIÓN GRÁFICA Y ACCESO A DATOS



1º CFGS DAW

Curso 2024-25

Introducción a JavaFX

8.1. Introducción a las aplicaciones JavaFX y Scene Builder

JavaFX fue inicialmente desarrollado y mantenido por *Oracle* a partir de *Java 8* (2014), pero a partir de *Java 11*, *Oracle* descontinuó su soporte y lo separó del *JDK*. Desde entonces, JavaFX se ha convertido en un proyecto de código abierto bajo el nombre de *OpenJFX*, y es mantenido por la comunidad.

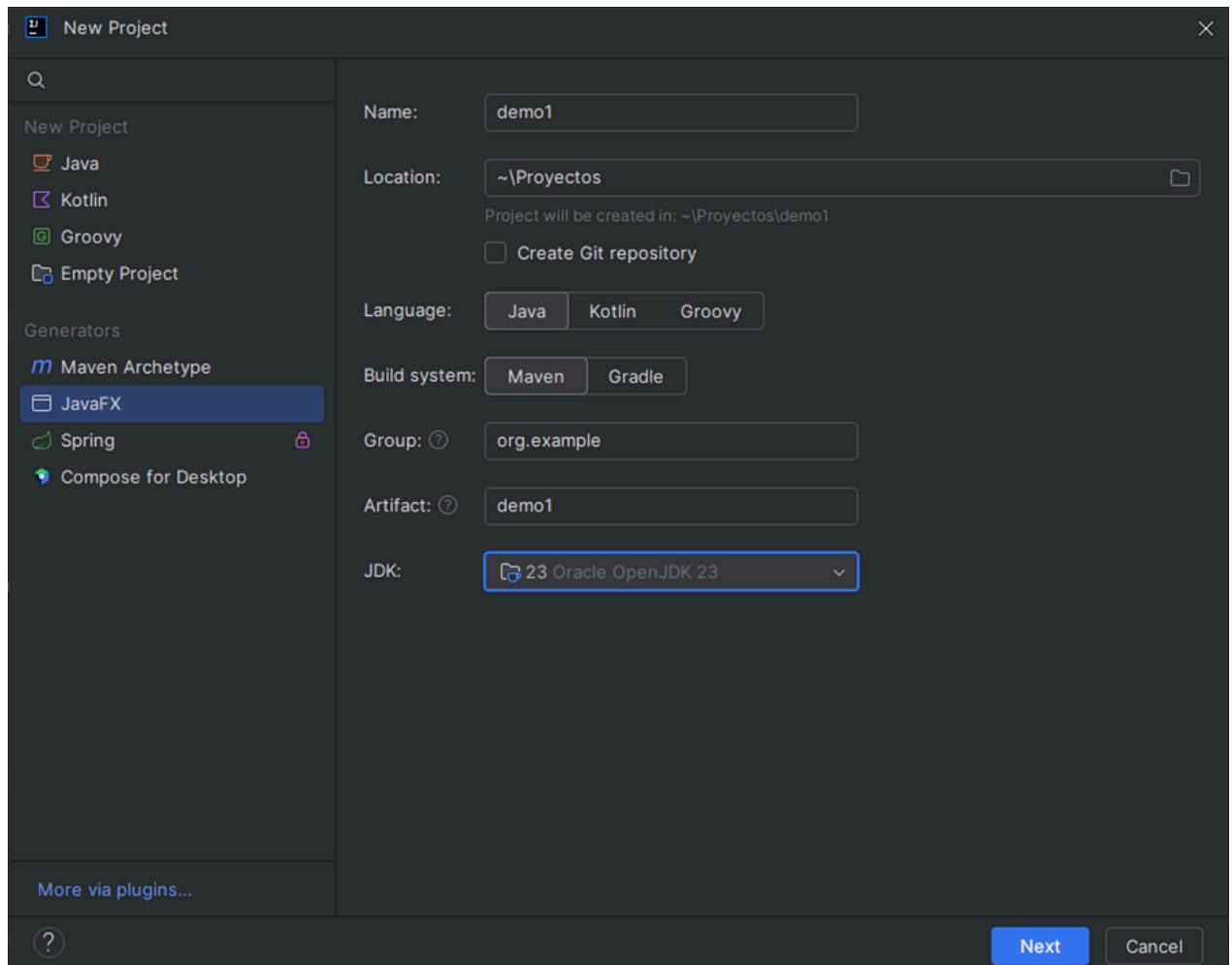
El código está disponible en el siguiente repositorio de GitHub: **openjdk/jfx: JavaFX mainline development <<https://github.com/openjdk/jfx>>**

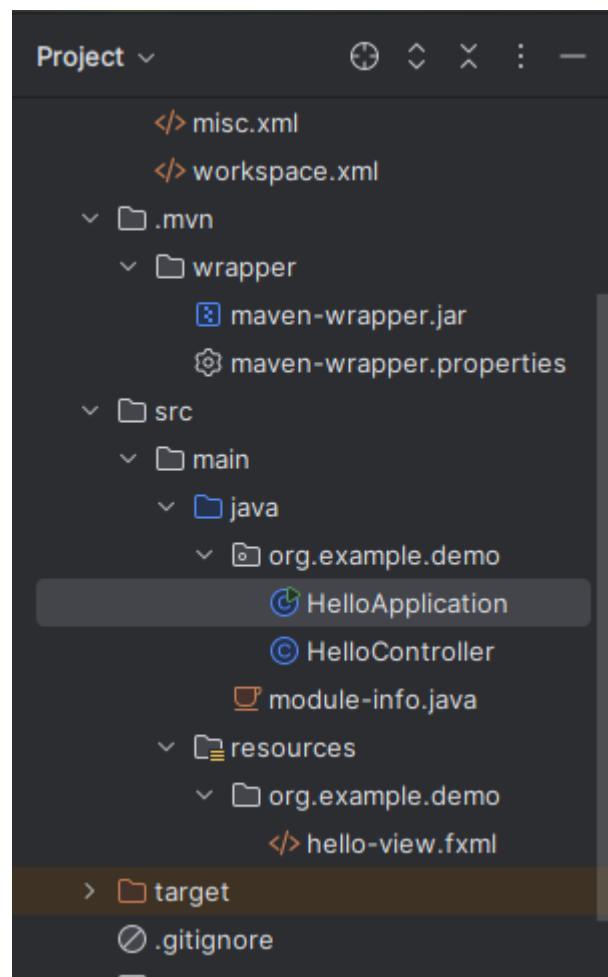
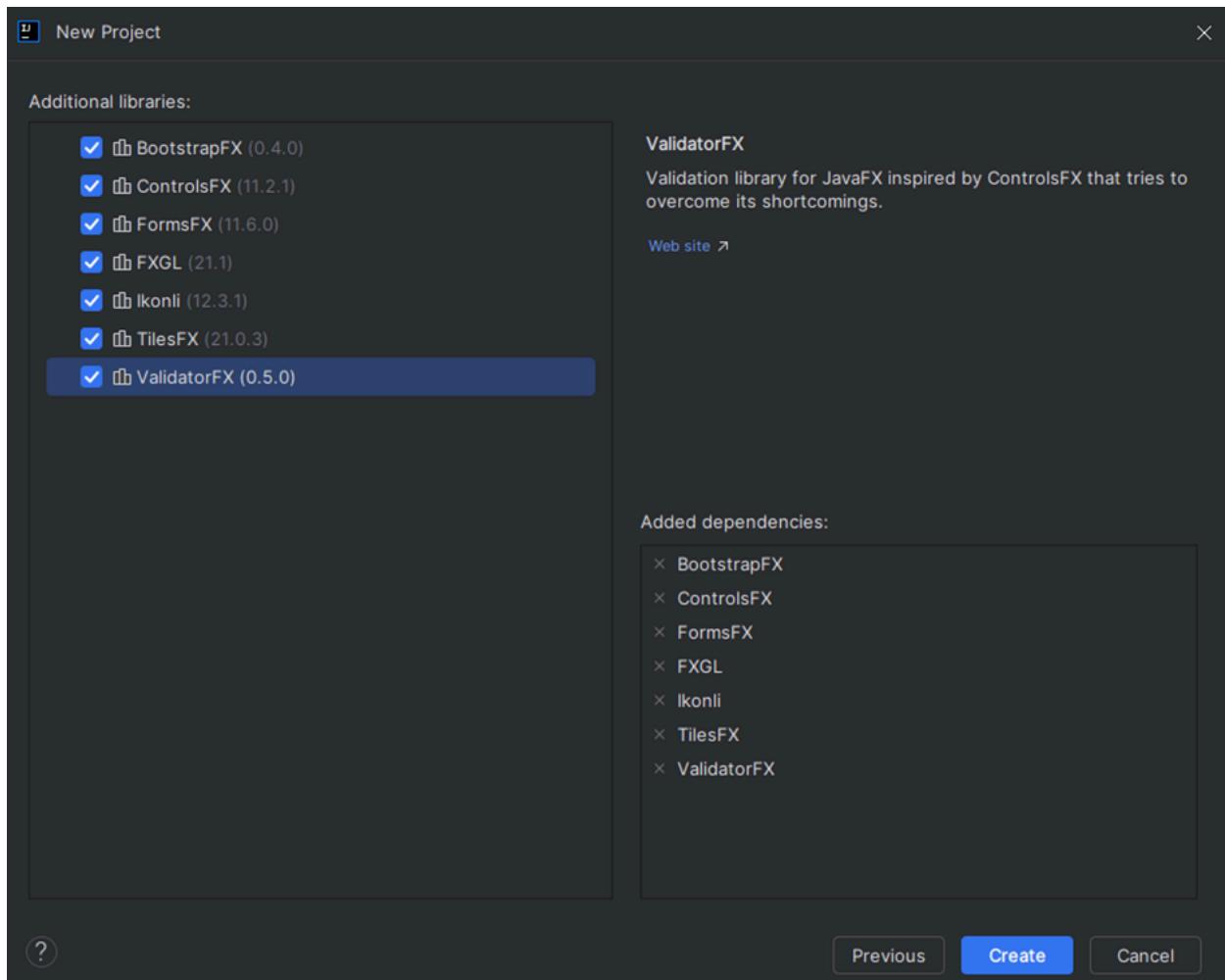
Y la **documentación oficial** la podemos consultar a través del siguiente enlace: **JavaFX <<https://openjfx.io/>>** junto a su **Javadoc <<https://openjfx.io/javadoc/24/>>**

8.1.1. Instalación

Para ponerlo todo a punto de forma fácil y rápida, utilizaremos los instaladores de *IntelliJ IDEA* como ayuda. Sigue los siguientes pasos:

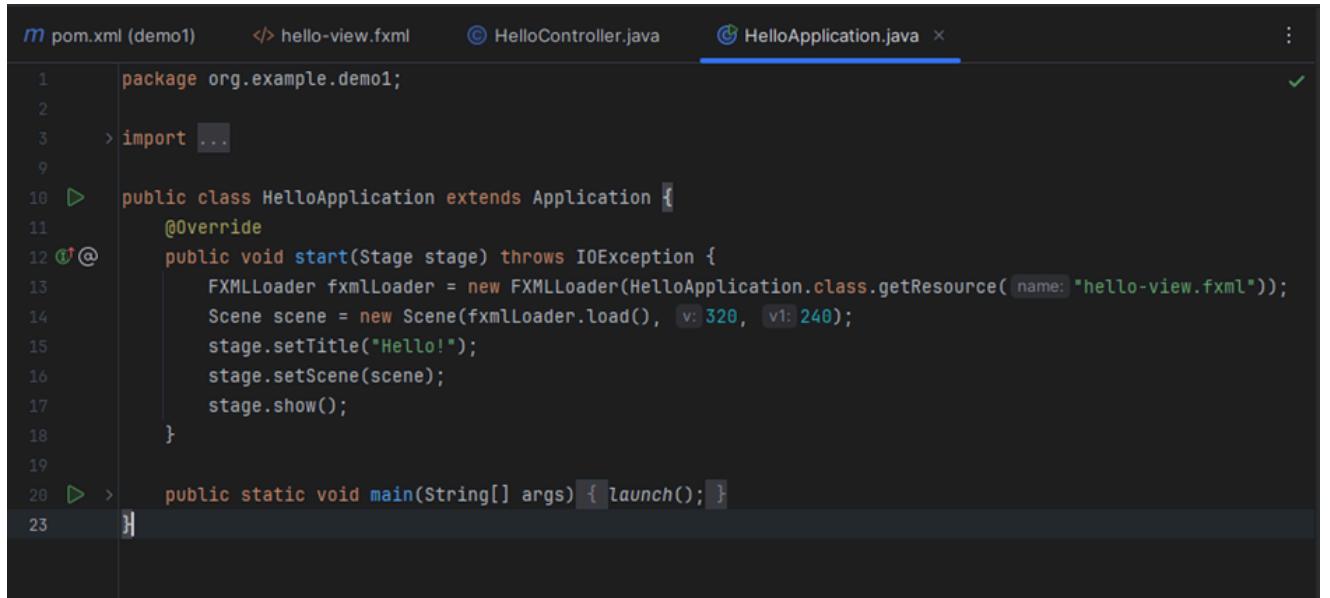
1. Crea un nuevo proyecto de tipo *JavaFX*,





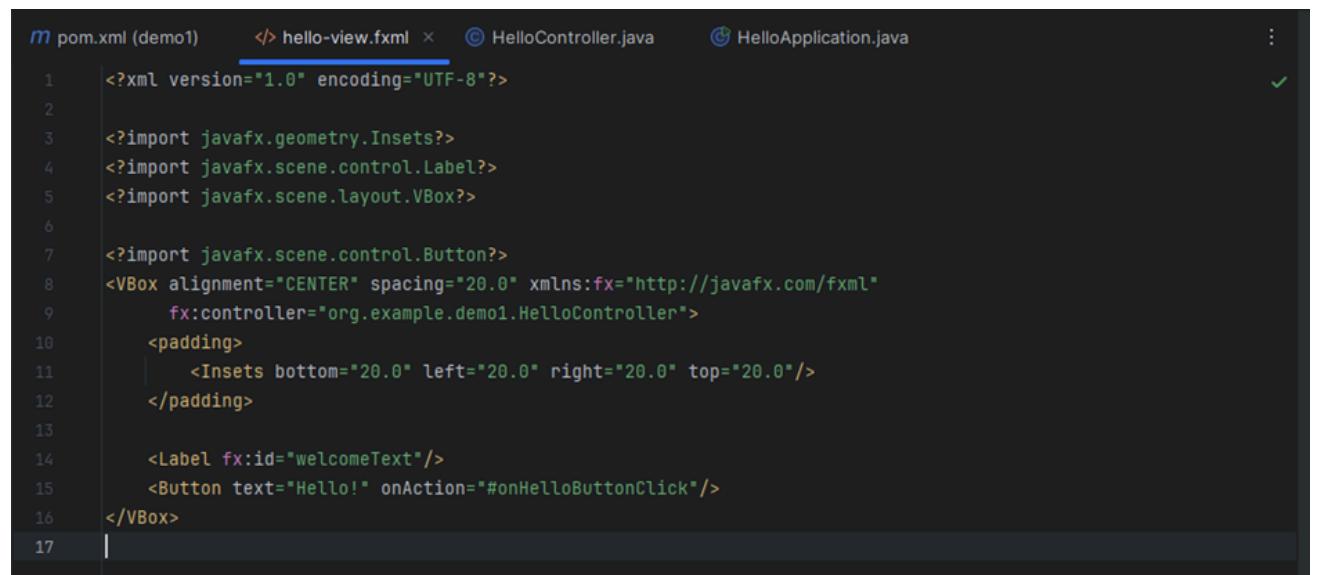
2. *OpenJFX* dispone de una API que proporciona clases e interfaces que son el núcleo para crear aplicaciones. Esta API está basada en MVC, que es el acrónimo de Model-View-Controller (Modelo-Vista-Controlador), un patrón de diseño de software que divide una aplicación en tres componentes interconectados. Este patrón es particularmente útil en aplicaciones con interfaces gráficas de usuario, como las desarrolladas con JavaFX.

La que vemos a continuación es la **clase principal (Stage)** → lanza la aplicación y carga el FXML (vista).



```
m pom.xml (demo1) </> hello-view.fxml © HelloController.java ⚡ HelloApplication.java × :  
1 package org.example.demo1;  
2  
3 > import ...  
4  
5 D public class HelloApplication extends Application {  
6     @Override  
7     public void start(Stage stage) throws IOException {  
8         FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));  
9         Scene scene = new Scene(fxmlLoader.load(), v: 320, v1: 240);  
10        stage.setTitle("Hello!");  
11        stage.setScene(scene);  
12        stage.show();  
13    }  
14  
15 D >     public static void main(String[] args) { launch(); }  
16 }  
17
```

FXML (Scene) → define la interfaz.

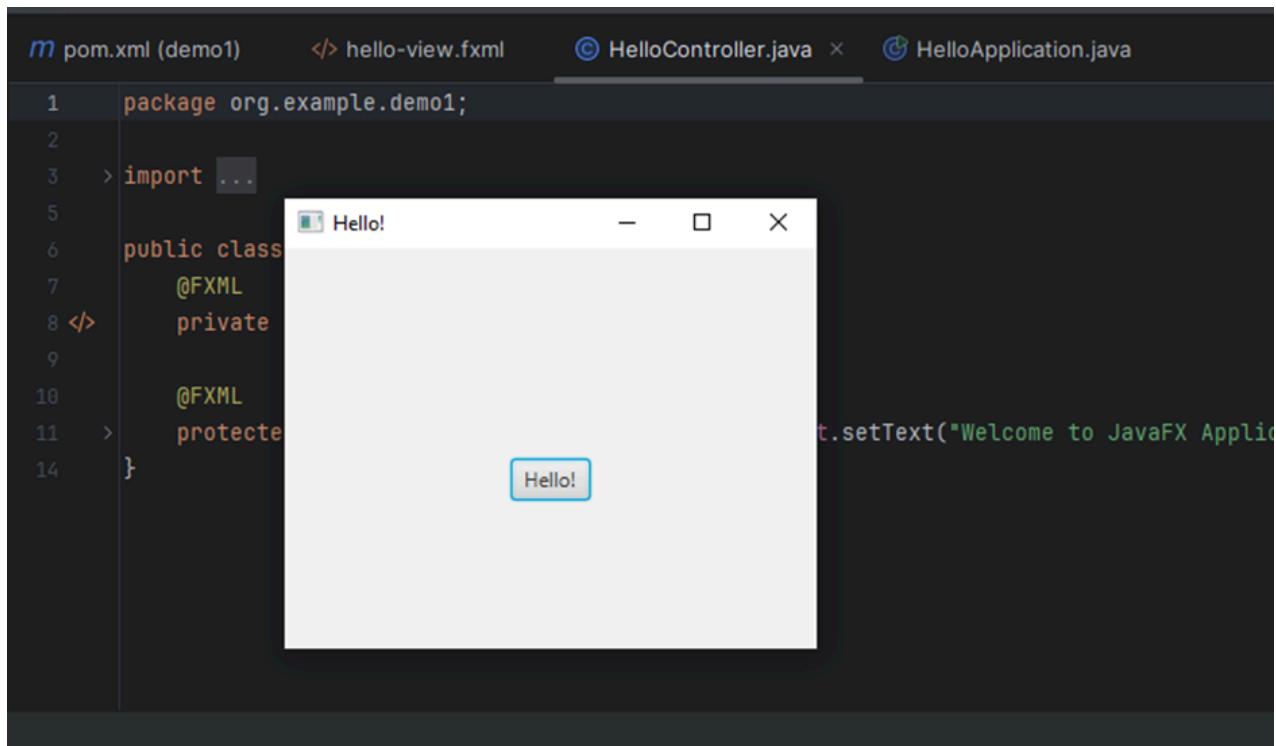


```
m pom.xml (demo1) </> hello-view.fxml × © HelloController.java ⚡ HelloApplication.java :  
1 <?xml version="1.0" encoding="UTF-8"?>  
2  
3 <?import javafx.geometry.Insets?>  
4 <?import javafx.scene.control.Label?>  
5 <?import javafx.scene.layout.VBox?>  
6  
7 <?import javafx.scene.control.Button?>  
8 <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"  
9     fx:controller="org.example.demo1.HelloController">  
10    <padding>  
11        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>  
12    </padding>  
13  
14    <Label fx:id="welcomeText"/>  
15    <Button text="Hello!" onAction="#onHelloButtonClick"/>  
16 </VBox>  
17 |
```

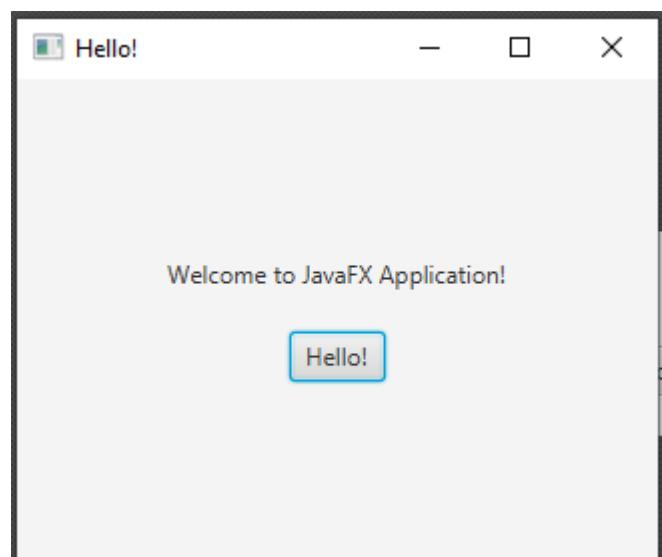
Y nos queda el **Controlador Java (Node)** → la clase que maneja los eventos (clics, acciones...). Es el intermediario entre el modelo y la vista. Siempre debe estar asociado a un archivo *FXML*. Utiliza anotaciones para enlazar los componentes de la interfaz de usuario con variables y métodos.

```
m pom.xml (demo1) </> hello-view.fxml <@> HelloController.java <@> HelloApplication.java : OFF
1 package org.example.demo1;
2
3 > import ...
4
5
6 public class HelloController {
7     @FXML
8 </>     private Label welcomeText;
9
10    @FXML
11 >     protected void onHelloButtonClick() { welcomeText.setText("Welcome to JavaFX Application!"); }
12
13 }
```

3. Si ejecutamos la aplicación, el proyecto de ejemplo que nos proporciona *IntelliJ IDEA* tiene la siguiente forma:



Si clickamos en el botón, aparecerá el texto dándonos la bienvenida a JavaFX:



Instalar *Scene Builder*

Descárgalo desde:

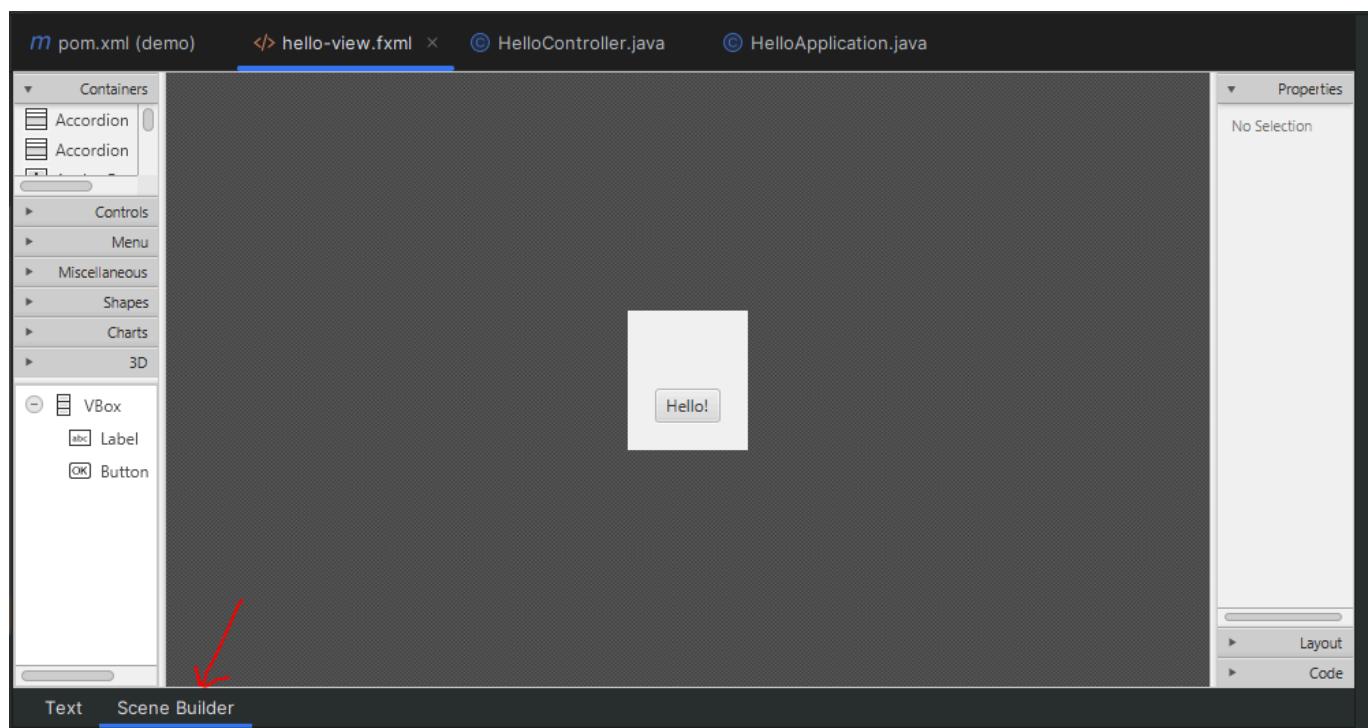
👉 <https://gluonhq.com/products/scene-builder/>
<https://gluonhq.com/products/scene-builder/>

e instálalo como cualquier programa.

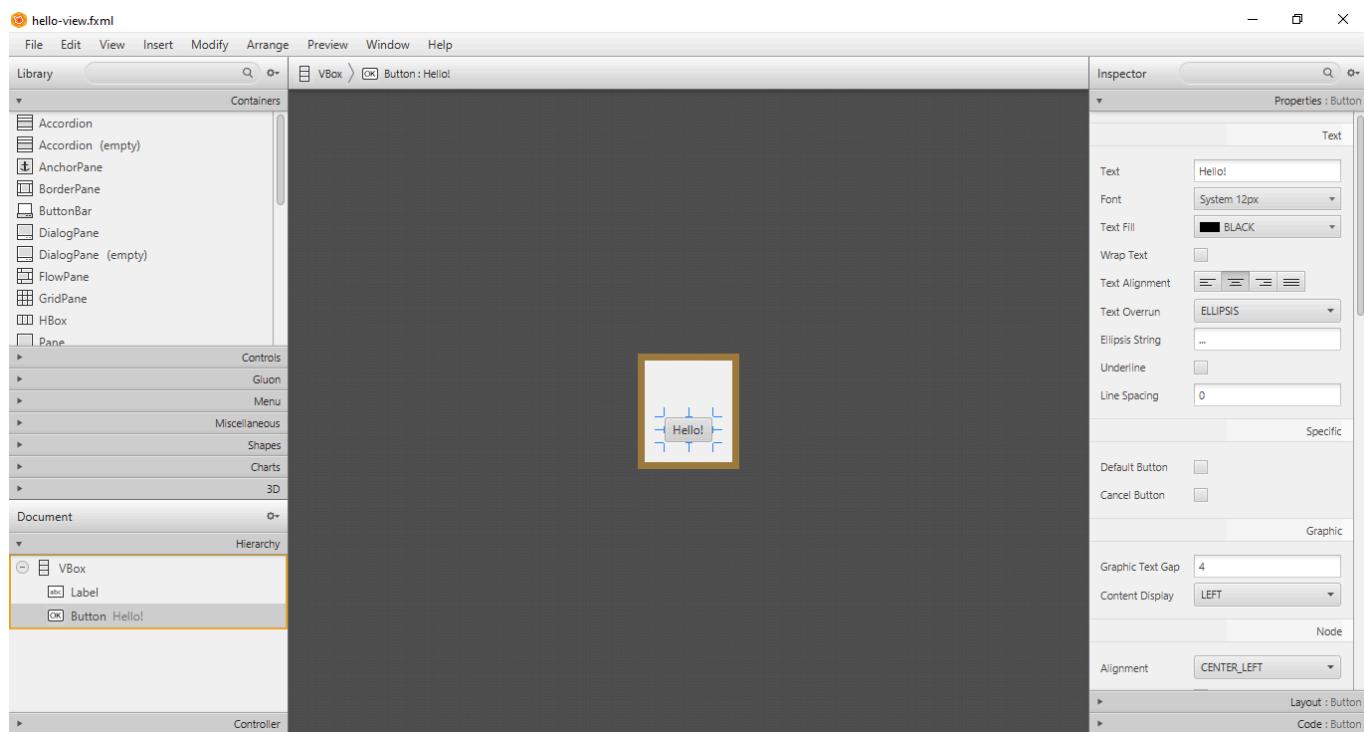
Al finalizar, anota la ruta del ejecutable, por ejemplo: *C:\Program Files\SceneBuilder\SceneBuilder.exe*

- Ve a *File > Settings*.
- Busca *Languages & Frameworks > JavaFX*
- En el campo "*Path to SceneBuilder*", pega la ruta del ejecutable que acabas de copiarte.

A partir de ese momento, quedará habilitada la vista más amigable del *FXML* desde la pestaña *Scene Builder*.



Aunque si lo prefieres, puedes usar directamente la aplicación desde fuera del IDE (suele ser más fluida):



8.1.2. Conociendo el entorno y primeros ejemplos

Para dar nuestros primeros pasos en JavaFX iremos a lo más fácil y básico. Vamos a dirigirnos a la vista (archivo `.fxml`), y observamos que tenemos dos componentes: una etiqueta (*Label*) y un botón (*Button*):

```
1 | <?xml version="1.0" encoding="UTF-8"?>
2 |
3 | <?import javafx.geometry.*?>
4 | <?import javafx.scene.control.*?>
5 | <?import javafx.scene.layout.*?>
6 |
7 | <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml">
8 |     <padding>
9 |         <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
10 |     </padding>
11 |
12 |     <Label fx:id="welcomeText" />
13 |     <Button onAction="#onHelloButtonClick" text="Hello!" textAlignment=
14 | </VBox>
```

Cada componente define unas características. Vamos a centrarnos primero en el botón.

1. Botones.

La propiedad *text* hace referencia al texto que aparecerá en el botón. Por otro lado, en este caso se aplica además la propiedad *textAlignment* para centrarlo, y nos queda la propiedad más importante: *onAction*, que es en la que nos vamos a parar.

La propiedad *onAction* define el nombre del método que se lanzará cuando hagamos *click* en el botón. Este método deberemos escribirlo en el controlador del *fxml*, que en nuestro caso es *HelloController*. Vamos a verlo.

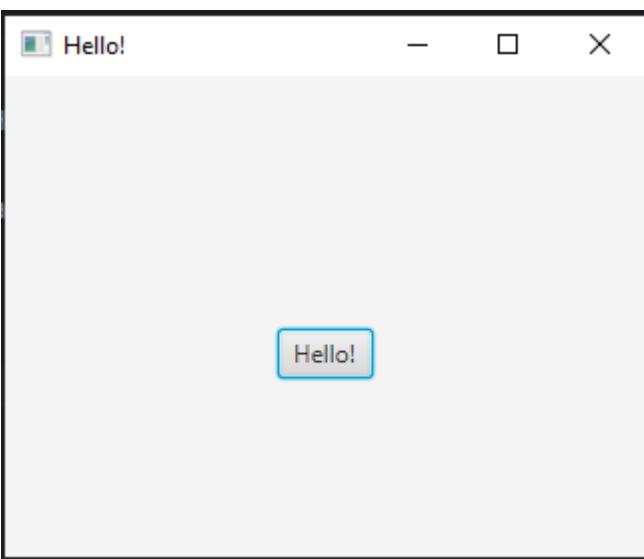
```
1 | import javafx.fxml.FXML;
2 | import javafx.scene.control.Label;
3 |
4 | public class HelloController {
5 |     @FXML
6 |     private Label welcomeText;
7 | }
```

```

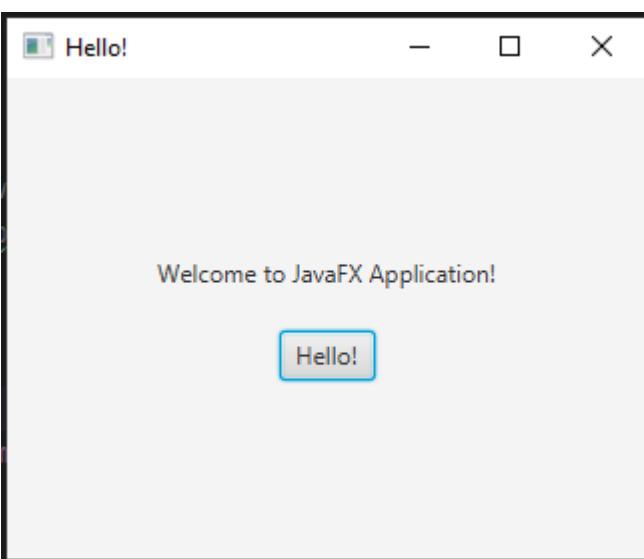
8     @FXML
9     protected void onHelloButtonClick() {
10         welcomeText.setText("Welcome to JavaFX Application!");
11     }
12 }
```

Por un lado, el controlador define una variable de tipo etiqueta (*Label*). El cuerpo del método del botón que acabamos de mencionar hace uso de dicha variable, que representa a la etiqueta, y le aplica el método *setText* para que en cuanto hagamos *click* se lance el evento y se modifique el texto.

Antes de hacer *click*:



Después de hacer *click*:



Esta forma de lanzar eventos con botones tiene otra variedad, que no necesariamente tiene que definir un método para realizar acciones sobre él. Para probar esta segunda opción, crearemos un nuevo botón en nuestro fxml:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.geometry.*?>
4 <?import javafx.scene.control.*?>
5 <?import javafx.scene.layout.*?>
6
7 <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml">
8     <padding>
9         <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
10    </padding>
11
12    <Label fx:id="welcomeText" />
13    <Button onAction="#onHelloButtonClick" text="Hello!" textAlignment="CENTER" />
14    <Button fx:id="boton" text="Pulsa!" textAlignment="CENTER" />
15 </VBox>
```

En este caso, le modificamos el texto y borramos la propiedad *onAction*. Además, le creamos un *id*, al que haremos referencia definiendo una variable de tipo *Button* desde el controlador. Vamos a ello:

```
1 import javafx.fxml.FXML;
2 import javafx.scene.control.Button;
3 import javafx.scene.control.Label;
4
5 public class HelloController {
6     @FXML
7     private Label welcomeText;
8     @FXML
9     private Button boton;
10
11    @FXML
12    protected void onHelloButtonClick() {
13        welcomeText.setText("Welcome to JavaFX Application!");
14    }
15 }
```

Ahora, crearemos el método *initialize()*. Este método se lanza automáticamente cuando iniciamos la aplicación, y estará escuchando constantemente para que cuando se produzca algún evento, se lance.

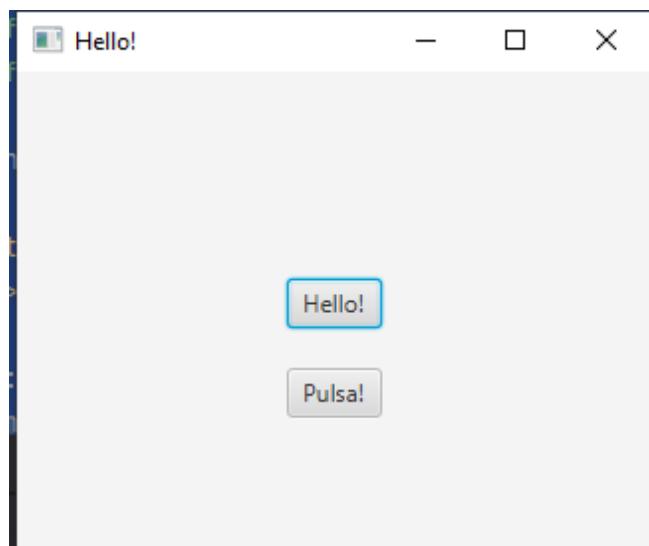
```

1 import javafx.fxml.FXML;
2 import javafx.scene.control.Button;
3 import javafx.scene.control.Label;
4
5 public class HelloController {
6     @FXML
7     private Label welcomeText;
8     @FXML
9     private Button boton;
10
11    @FXML
12    public void initialize(){
13        boton.setOnAction(e -> System.out.println("Hola"));
14    }
15
16    @FXML
17    protected void onHelloButtonClick() {
18        welcomeText.setText("Welcome to JavaFX Application!");
19    }
20}

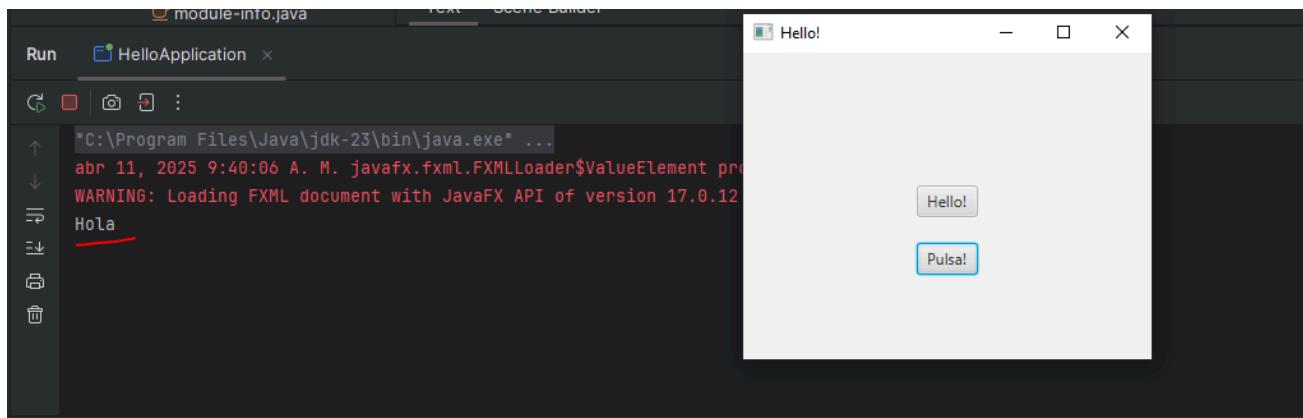
```

En nuestro caso, al botón le hemos creado un evento *setOnAction*, que imprimirá "Hola" en la consola cuando hagamos *click* en él.

Vamos a probarlo:



Y si hacemos *click* en "Pulsa!":



Ejemplo. Contador que aumenta cada vez que se pulsa un botón

Supongamos un simple contador. Reutilizaremos el botón "*Pulsa!*" creado en el ejemplo anterior.

Modificaremos las propiedades de dicho botón desde el *fxml* para adaptarlo al contador:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.geometry.*?>
4 <?import javafx.scene.control.*?>
5 <?import javafx.scene.layout.*?>
6
7 <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/
8     <padding>
9         <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
10    </padding>
11
12    <Label fx:id="welcomeText" />
13    <Button onAction="#onHelloButtonClick" text="Hello!" textAlignme
14        <Button fx:id="contador" text="Contar" textAlignment="CENTER" />
15    </VBox>
```

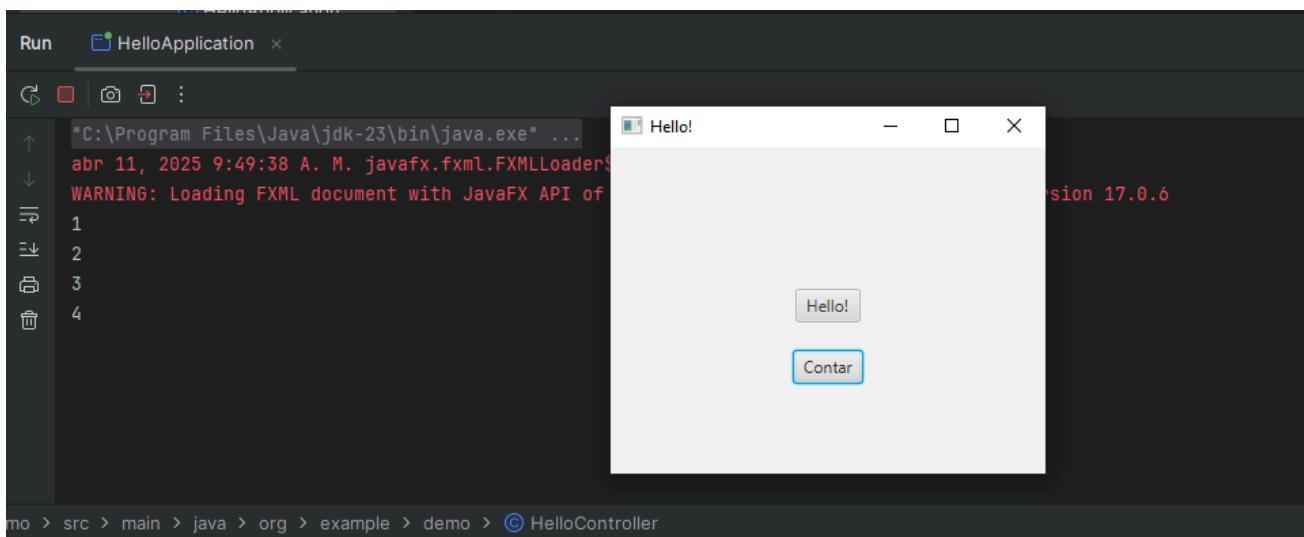
Desde el controlador también deberemos modificar la referencia que teníamos, ya que ahora el id de nuestro botón es "*contador*". Además, ahora por consola ya no imprimiremos "Hola", sino que tendremos que contar: 1, 2, 3, 4,... Para ello, crearemos también una variable estática en la clase. Quedaría de la siguiente manera:

```

1 import javafx.fxml.FXML;
2 import javafx.scene.control.Button;
3 import javafx.scene.control.Label;
4
5 public class HelloController {
6
7     static int cont;
8
9     @FXML
10    private Label welcomeText;
11
12    @FXML
13    private Button contador;
14
15    @FXML
16    public void initialize(){
17        contador.setOnAction(e -> System.out.println(++cont));
18    }
19
20    @FXML
21    protected void onHelloButtonClick() {
22        welcomeText.setText("Welcome to JavaFX Application!");
23    }
24}

```

Lo probamos:



Para no depender de la consola, iremos imprimiendo el valor del contador también en la pantalla. Introduciremos una nueva etiqueta en el *FXML* para ello:

```

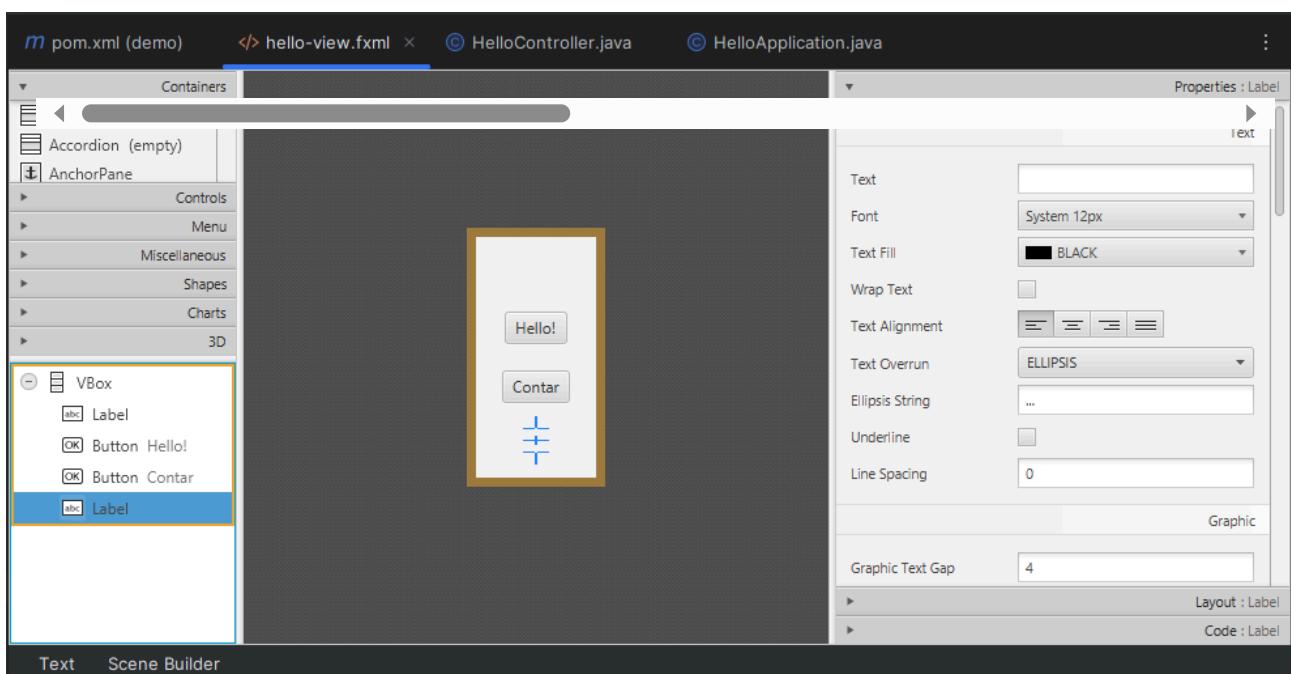
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.*?>

```

```

4 <?import javafx.scene.control.*?>
5 <?import javafx.scene.layout.*?>
6
7 <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/
8     <padding>
9         <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
10    </padding>
11
12    <Label fx:id="welcomeText" />
13    <Button onAction="#onHelloButtonClick" text="Hello!" textAlignme
14        <Button fx:id="contador" text="Contar" textAlignment="CENTER" />
15        <Label fx:id="contando" />
16 </VBox>

```



Nos toca modificar el controlador de nuevo. Ahora, el método `setOnAction` del botón no va a realizar un `println`, sino un `setText` en la nueva etiqueta:

```

package org.example.demo;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

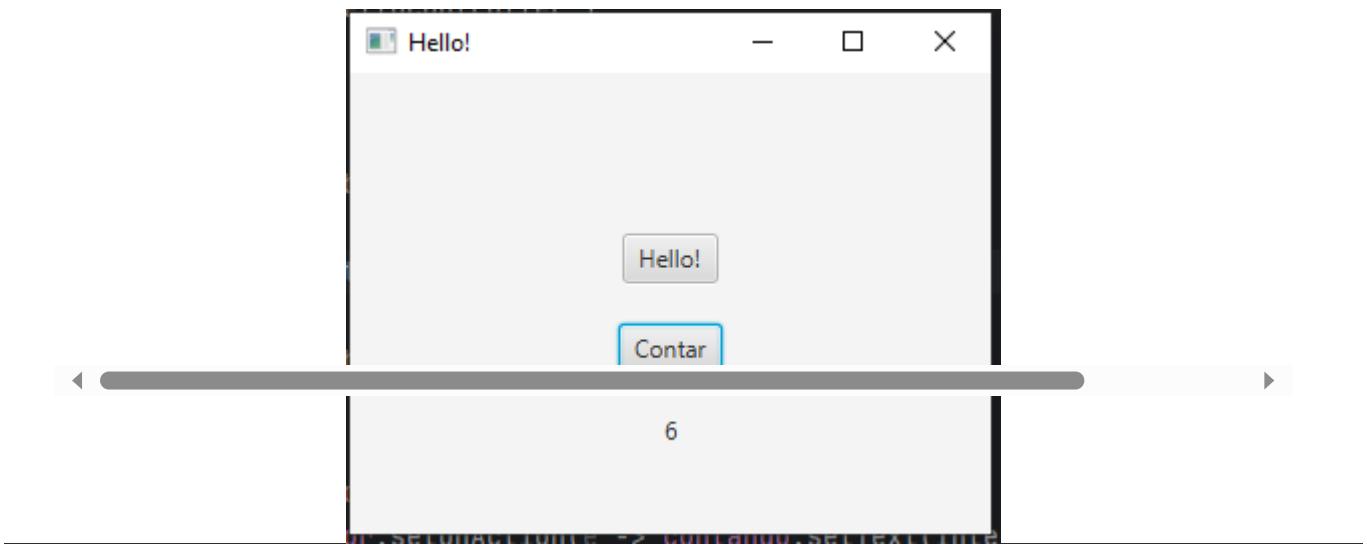
public class HelloController {

    static int cont;
    @FXML

```

```
12     private Label welcomeText;
13     @FXML
14     private Button contador;
15     @FXML
16     private Label contando;
17
18     @FXML
19     public void initialize(){
20         contador.setOnAction(e -> contando.setText(Integer.toString(
21     })
22
23     @FXML
24     protected void onHelloButtonClick() {
25         welcomeText.setText("Welcome to JavaFX Application!");
26     }
}
```

Lo probamos de nuevo:



8.1.3. Objetos (modelo)

Hasta ahora, hemos hablado del *Modelo-Vista-Controlador*, pero no hemos identificado eso del "modelo". Representa la lógica de negocio y los datos en la aplicación, es decir, las clases, interfaces, etc., que tenemos en nuestro proyecto.

Como todavía no tenemos ninguno, vamos a crearlo. Para seguir con el ejemplo del contador, crearemos una clase que represente lo que necesitamos:

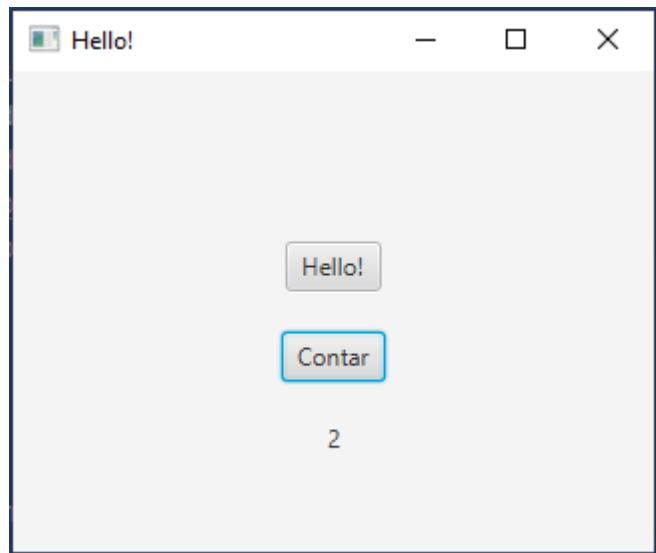
```
1 public class Contador {  
2  
3     private int contador;  
4  
5     public Contador(){  
6  
7         }  
8  
9     public void contar(){  
10        contador++;  
11    }  
12  
13     public int getContador(){  
14         return contador;  
15     }  
16  
17 }
```

Este modelo deberemos integrarlo con nuestro controlador, por lo que eliminaremos la variable estática que teníamos para contar y en *initialize()* crearemos un objeto *Contador* que iremos manipulando cada vez que ejecutemos el método *.setOnAction* del botón que servía para contar:

```
1 import javafx.fxml.FXML;  
2 import javafx.scene.control.Button;  
3 import javafx.scene.control.Label;  
4  
5 public class HelloController {  
6  
7     @FXML  
8     private Label welcomeText;
```

```
9      @FXML
10     private Button contador;
11
12     @FXML
13     private Label contando;
14
15     @FXML
16     public void initialize(){
17         Contador cont = new Contador();
18         contador.setOnAction(e -> {
19             cont.contar();
20             contando.setText(Integer.toString(cont.getContador()));
21         });
22     }
23
24     @FXML
25     protected void onHelloButtonClick() {
26         welcomeText.setText("Welcome to JavaFX Application!");
27     }
28 }
```

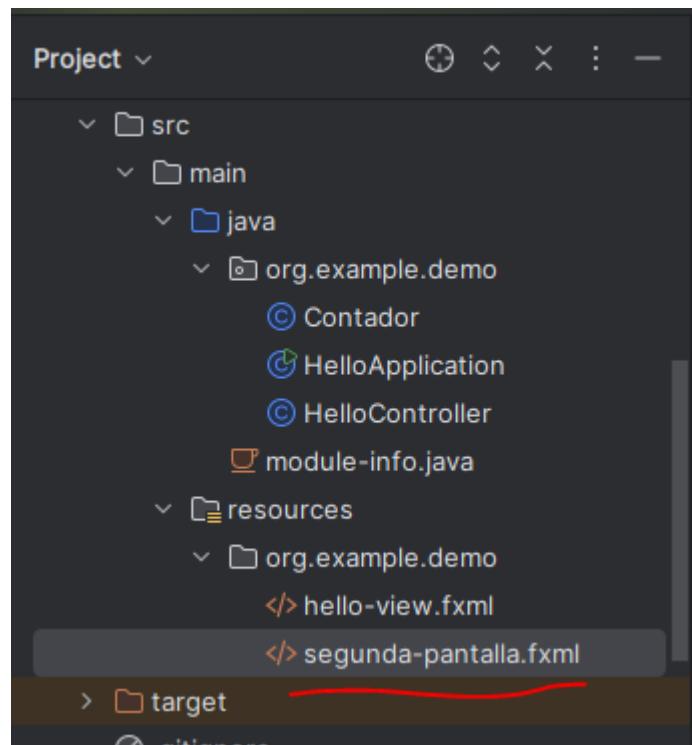
Lánzalo y comprueba que el comportamiento de la aplicación sigue siendo el mismo.



8.1.4. Moverse entre pantallas (Scene)

Hasta ahora hemos estado toqueteando una sola pantalla, pero, ¿cómo podemos movernos entre pantallas en caso de tener más de una?

Pues para empezar, necesitaremos tener en nuestro proyecto más de una vista, así que vamos a crear un nuevo *fxml* llamado "*segunda-pantalla.fxml*" en la carpeta *resources*:

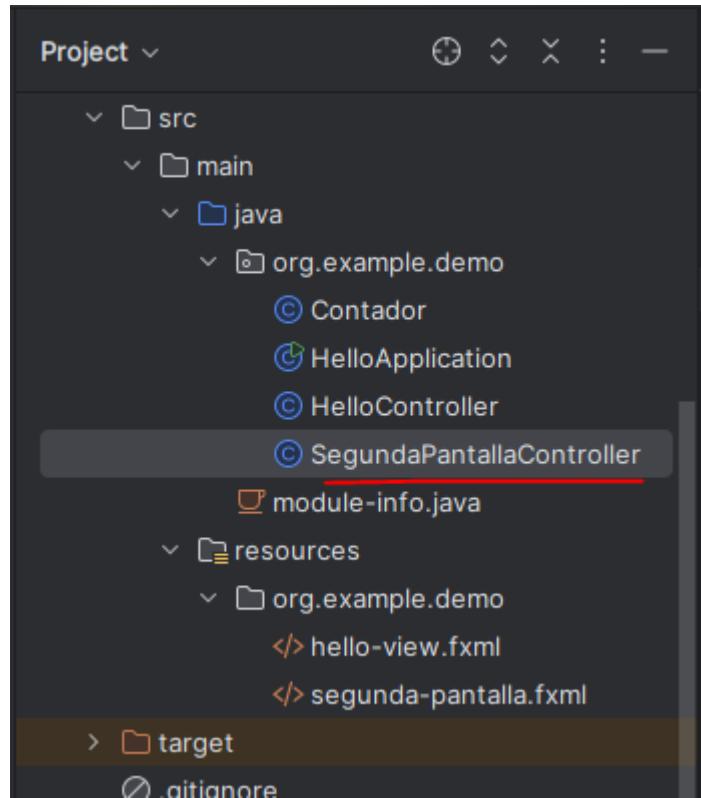


Al crearla y abrirse, empezará a quejarse porque no le hemos indicado qué controlador se hará cargo de esta pantalla:

```
</> hello-view.fxml x  Ⓜ HelloController.java    </> segunda-pantalla.fxml x  Ⓜ Contador.java    Ⓜ HelloApplication.java :  
4  <?import java.util.*?>  
5  <?import javafx.scene.*?>  
6  <?import javafx.scene.control.*?>  
7  <?import javafx.scene.layout.*?>  
8  
9  <AnchorPane xmlns="http://javafx.com/javafx"  
10   xmlns:fx="http://javafx.com/fxml"  
11   fx:controller="org.example.demo.SegundaPantalla"  
12   prefHeight="400.0" prefWidth="600.0">  
13  
14  </AnchorPane>  
15 |
```

The screenshot shows the IntelliJ IDEA code editor with several tabs open: 'hello-view.fxml', 'HelloController.java', 'segunda-pantalla.fxml' (which is currently active), 'Contador.java', and 'HelloApplication.java'. The 'segunda-pantalla.fxml' tab has a blue underline. The code in the editor is an FXML snippet defining an AnchorPane with a controller binding to 'SegundaPantalla'. The status bar at the bottom shows 'Text' and 'Scene Builder'.

Vamos a crearlo también:



```
1 package org.example.demo;
2
3 public class SegundaPantallaController {
```

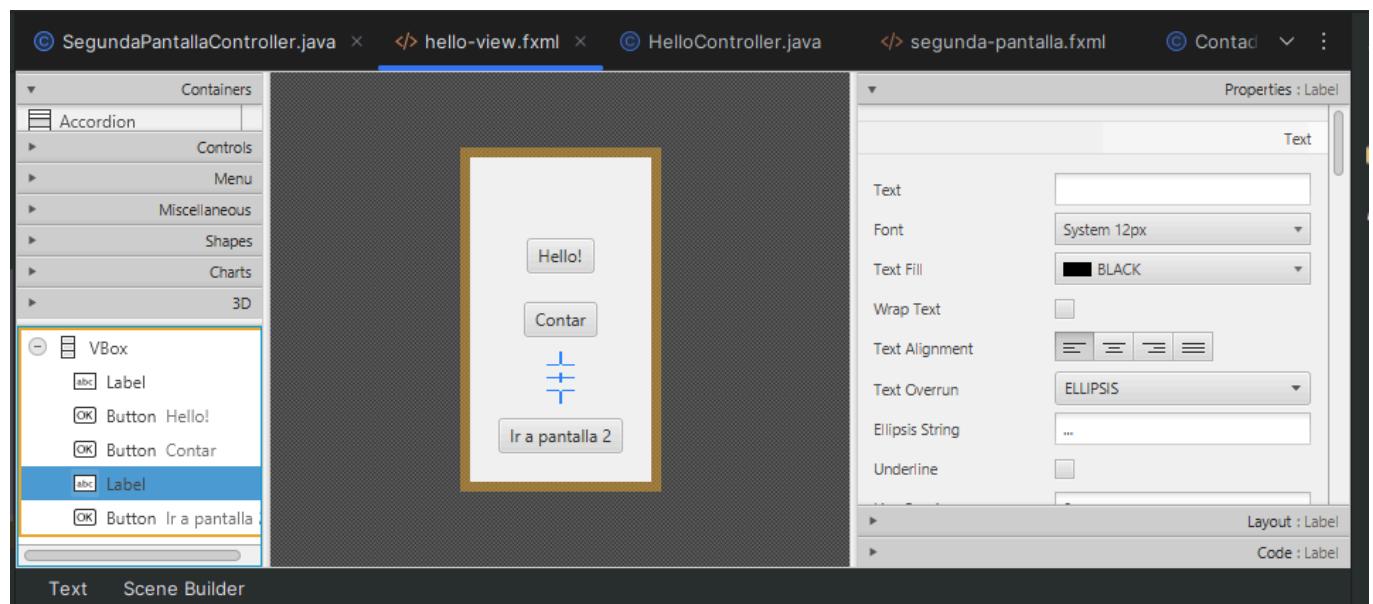
Ya podemos indicarle al fxml que este va a ser su controlador:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import java.lang.*?>
4 <?import java.util.*?>
5 <?import javafx.scene.*?>
6 <?import javafx.scene.control.*?>
7 <?import javafx.scene.layout.*?>
8
9 <AnchorPane xmlns="http://javafx.com/javafx"
10           xmlns:fx="http://javafx.com/fxml"
11           fx:controller="org.example.demo.SegundaPantallaController"
12           prefHeight="400.0" prefWidth="600.0">
13
14 </AnchorPane>
15
```

Introduciremos un nuevo botón en cada uno de los *fxml* que servirá para ir de una pantalla a otra.

- Pantalla *hello-view*:

```
1 | <?xml version="1.0" encoding="UTF-8"?>
2 |
3 | <?import javafx.geometry.*?>
4 | <?import javafx.scene.control.*?>
5 | <?import javafx.scene.layout.*?>
6 |
7 | <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxm
8 |     <padding>
9 |         <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
10 |     </padding>
11 |
12 |     <Label fx:id="welcomeText" />
13 |     <Button onAction="#onHelloButtonClick" text="Hello!" textAlignment=
14 |     <Button fx:id="contador" text="Contar" textAlignment="CENTER" />
15 |     <Label fx:id="contando" />
16 |     <Button text="Ir a pantalla 2" textAlignment="CENTER" onAction="#ir
17 | </VBox>
```

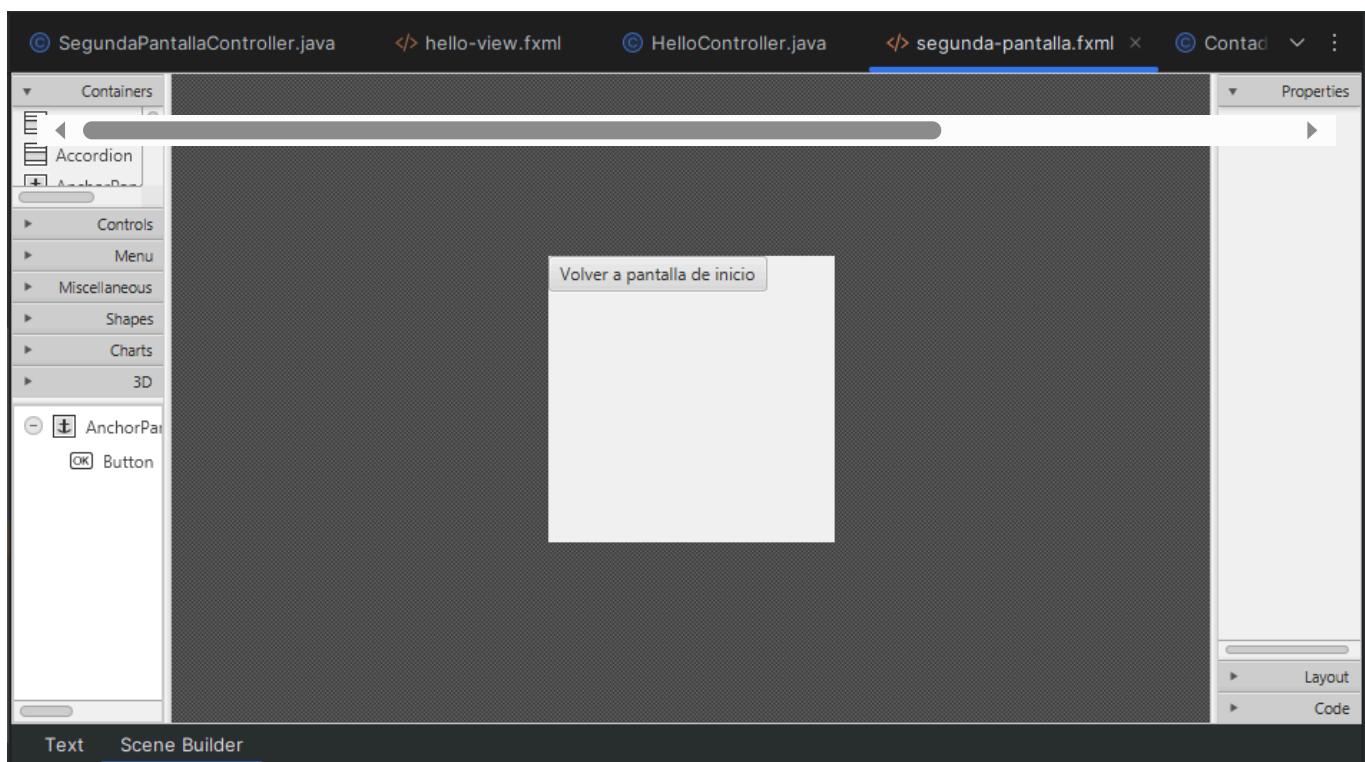


- Pantalla *segunda-pantalla*:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
```

```

5  <?import java.util.*?>
6  <?import javafx.scene.*?>
7  <?import javafx.scene.control.*?>
8  <?import javafx.scene.layout.*?>
9
10 <AnchorPane xmlns="http://javafx.com/javafx"
11   xmlns:fx="http://javafx.com/fxml"
12   fx:controller="org.example.demo.SegundaPantallaController"
13   prefHeight="200.0" prefWidth="200.0">
14
15   <Button text="Volver a pantalla de inicio" textAlignment="CENTER" o
16
</AnchorPane>
```



Ahora nos queda modificar los controladores. Pero antes, deberemos realizar unas adaptaciones en nuestra clase de la App principal, *HelloApplication.java*.

- La primera modificación que deberemos hacer es sacar la definición de la variable de tipo *Scene* que está dentro del *.start()*, para hacerla estática y accesible desde cualquier método de la clase:

```

package org.example.demo;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
```

```
5 import javafx.scene.Scene;
6 import javafx.stage.Stage;
7
8 import java.io.IOException;
9
10 public class HelloApplication extends Application {
11
12     private static Scene scene;
13
14     @Override
15     public void start(Stage stage) throws IOException {
16         FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.g
17         scene = new Scene(fxmlLoader.load(), 320, 240);
18         stage.setTitle("Hello!");
19         stage.setScene(scene);
20         stage.show();
21     }
22
23     public static void main(String[] args) {
24         launch();
25     }
26 }
```

- Ahora, crearemos el siguiente método nuevo:

```
1 private static Parent loadFXML(String fxml) throws IOException {
2     FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.g
3     return fxmlLoader.load();
4 }
```

Si te fijas, es lo mismo que hace el `.start()`, pero en versión reutilizable.

- Crearemos también este otro método, que llamará cada controlador para cambiar de pantalla cuando sea necesario.

```
1 static void setRoot(String fxml) throws IOException {
2     scene.setRoot(loadFXML(fxml));
3 }
```

Ahora sí, vamos a cada controlador a escribir el código necesario.

- Controlador *HelloController*:

```

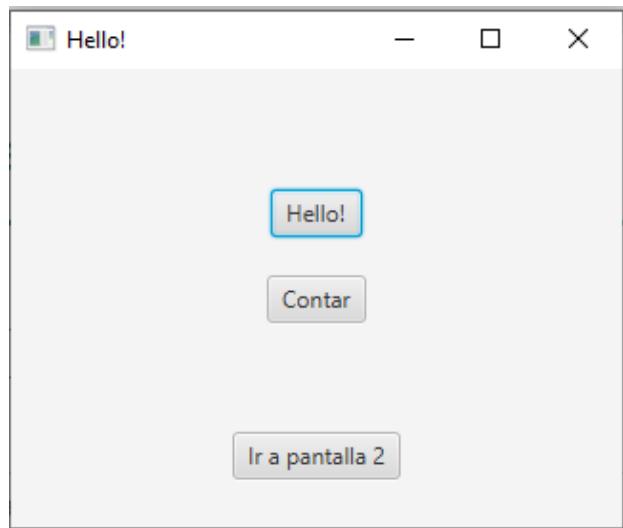
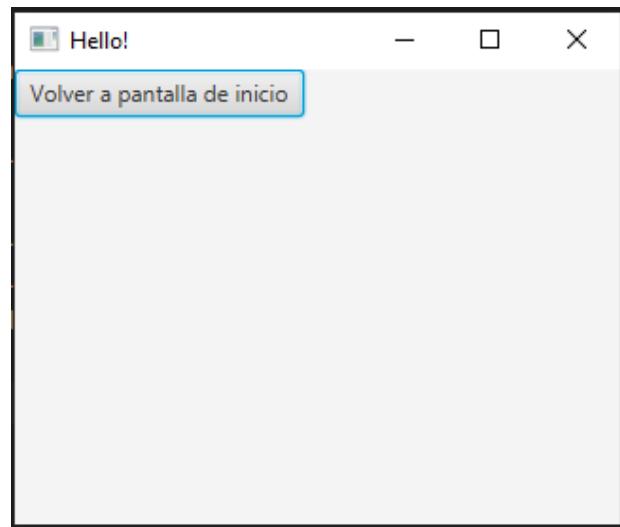
1 package org.example.demo;
2
3 import javafx.event.ActionEvent;
4 import javafx.fxml.FXML;
5 import javafx.scene.control.Button;
6 import javafx.scene.control.Label;
7
8 import java.io.IOException;
9
10 public class HelloController {
11
12     @FXML
13     private Label welcomeText;
14
15     @FXML
16     private Button contador;
17
18     @FXML
19     private Label contando;
20
21     @FXML
22     public void initialize(){
23         Contador cont = new Contador();
24         contador.setOnAction(e -> {
25             cont.contar();
26             contando.setText(Integer.toString(cont.getContador()));
27         });
28
29     }
30
31     @FXML
32     protected void onHelloButtonClick() {
33         welcomeText.setText("Welcome to JavaFX Application!");
34     }
35
36     @FXML
37     public void irAPantalla2() throws IOException {
38         HelloApplication.setRoot("segunda-pantalla");
39     }

```

- Controlador *SegundaPantallaController*:

```
1 package org.example.demo;  
2  
3 import javafx.fxml.FXML;  
4  
5 import java.io.IOException;  
6  
7 public class SegundaPantallaController {  
8  
9     @FXML  
10    public void irAPantallaHello() throws IOException {  
11        HelloApplication.setRoot("hello-view");  
12    }  
13}  
14}
```

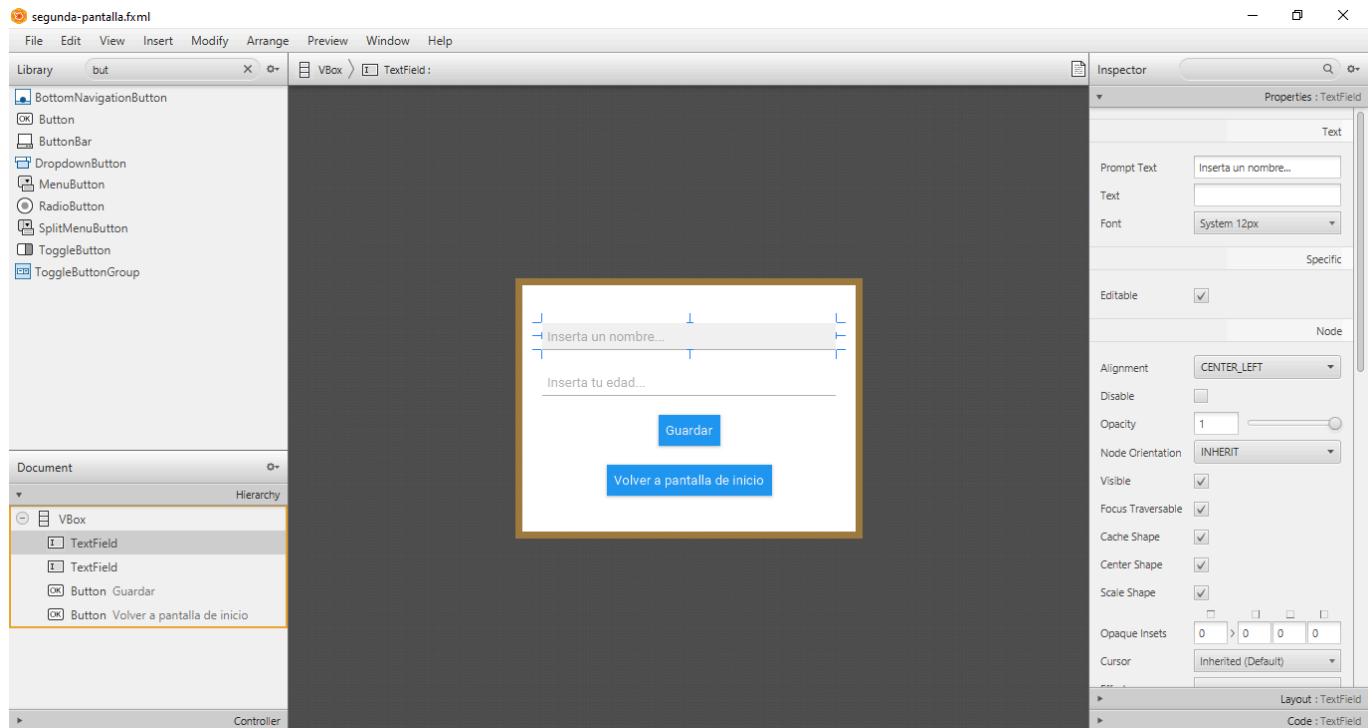
Ya podemos probarlo:



8.1.5. Manipular info desde formularios

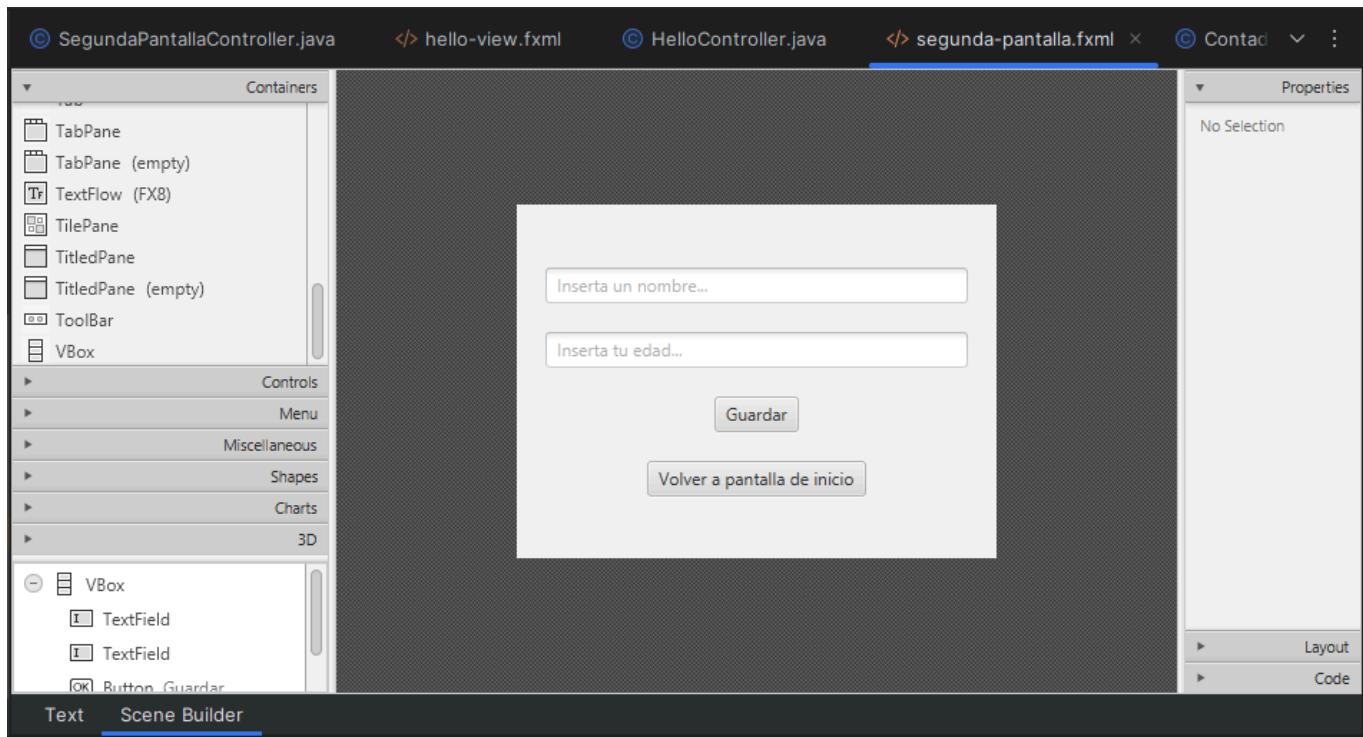
En *JavaFX*, para guardar los valores de un *TextField* (o cualquier otro componente de entrada de datos) como atributos de un objeto, simplemente tenemos que obtener el texto del *TextField* y pasarlo al objeto correspondiente.

Empezaremos insertando en nuestra segunda pantalla dos *TextFields*, que servirán para guardar el *nombre* y la *edad* de una Persona:



Además, insertaremos un botón "Guardar".

Volvemos a *IntelliJ*...



Primero retocamos el *fxml* para asignar a cada *TextField* un *id* y asignar una acción al botón de *Guardar*.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.geometry.Insets?>
4 <?import javafx.scene.control.Button?>
5 <?import javafx.scene.control.TextField?>
6 <?import javafx.scene.layout.VBox?>
7
8 <VBox alignment="CENTER" prefHeight="247.0" prefWidth="335.0" spacing="
9 <padding>
10 <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
11 </padding>
12 <TextField fx:id="nombreTextField" promptText="Inserta un nombre..." />
13 <TextField fx:id="edadTextField" promptText="Inserta tu edad..." />
14 <Button text="Guardar" onAction="#guardarPersona"/>
15 <Button onAction="#irAPantallaHello" text="Volver a pantalla de ini
16 </VBox>
```

A continuación, creamos nuestra clase **Persona** que será manipulada por el controlador:

```

1 package org.example.demo;
2
3 public class Persona {
```

```

4     private String nombre;
5     private int edad;
6
7     public Persona(String nombre, int edad) {
8         this.nombre = nombre;
9         this.edad = edad;
10    }
11
12    public String getNombre() {
13        return nombre;
14    }
15
16    public int getEdad() {
17        return edad;
18    }
19
20}

```

Y ahora sí que sí, nos dirigimos al controlador que usará la propiedad `.getText()` para guardarse lo que tengamos escrito en cada `TextField`, y creará una nueva `Persona` con esos valores de atributos *nombre* y *edad*.

```

package org.example.demo;

import javafx.fxml.FXML;
import javafx.scene.control.TextField;

import java.io.IOException;

public class SegundaPantallaController {

    @FXML
    private TextField nombreTextField;

    @FXML
    private TextField edadTextField;

    @FXML
    private void guardarPersona() {

        String nombre = nombreTextField.getText();
        int edad;

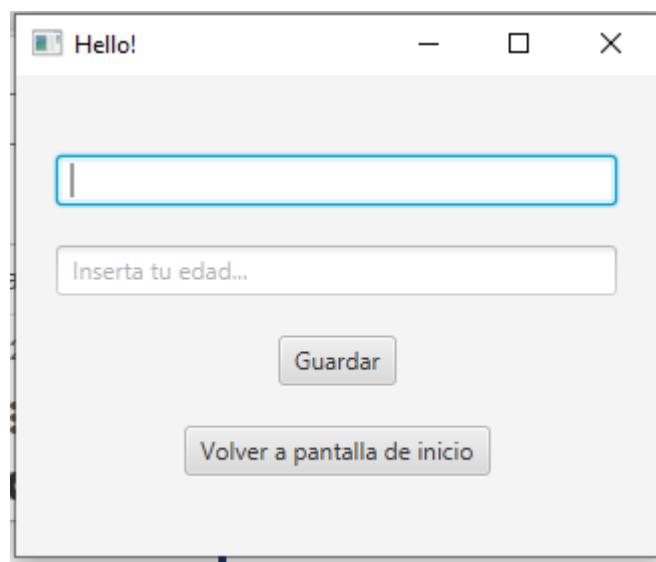
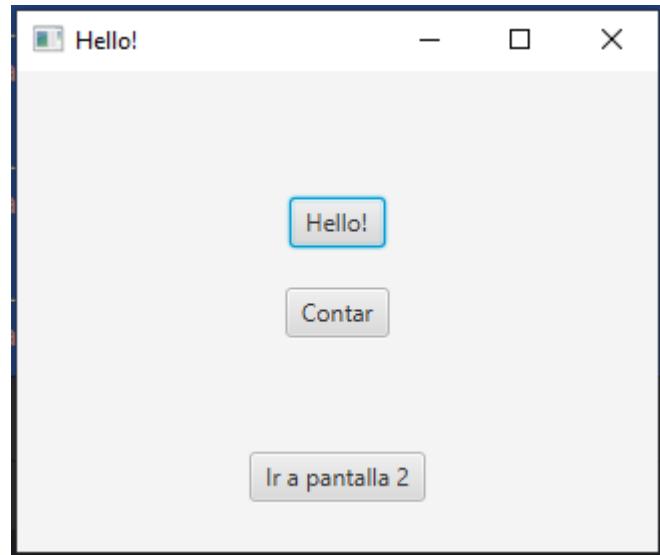
        try {

```

```
23         edad = Integer.parseInt(edadTextField.getText());
24     } catch (NumberFormatException e) {
25         System.out.println("Edad inválida");
26         return;
27     }
28
29     Persona persona = new Persona(nombre, edad);
30     System.out.println("Persona creada: " + persona.getNombre() + "
31 }
32
33 @FXML
34 public void irAPantallaHello() throws IOException {
35     HelloApplication.setRoot("hello-view");
36 }
37
38 }
```

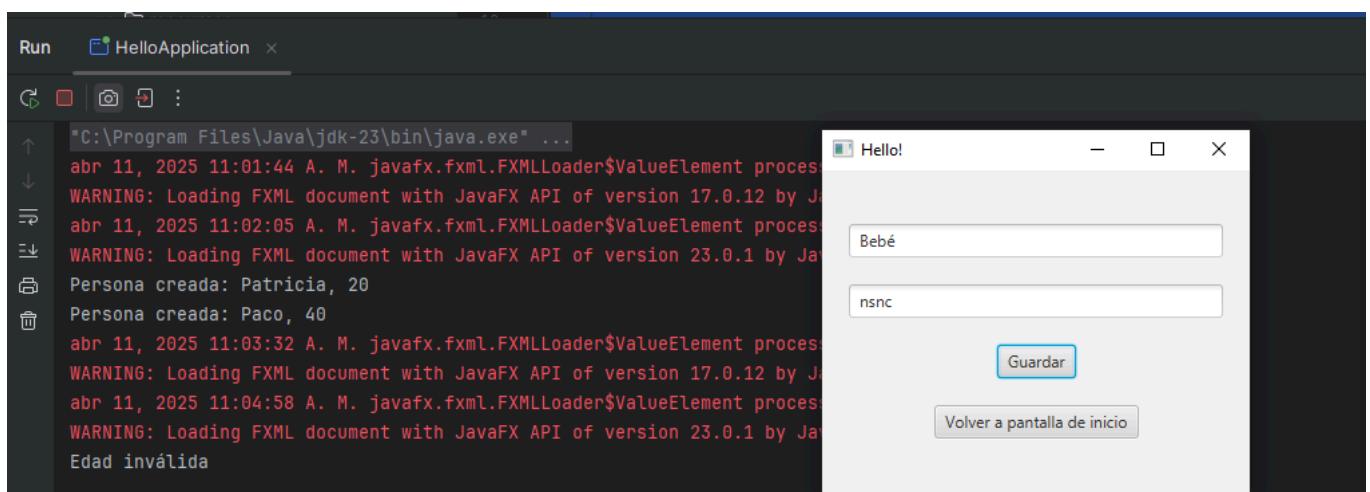
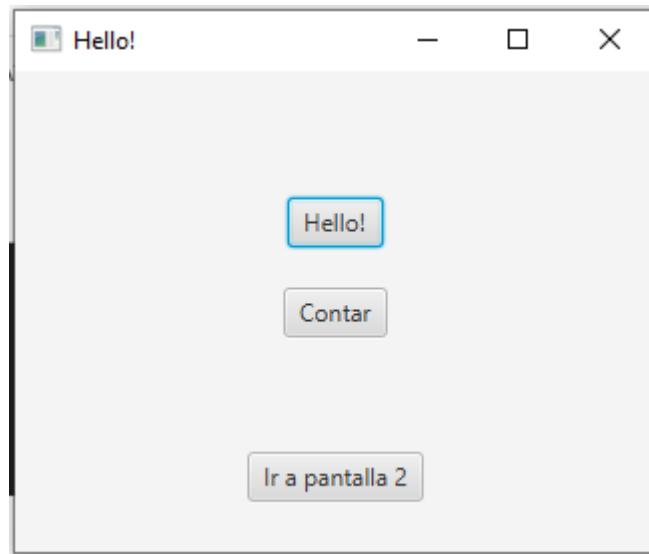
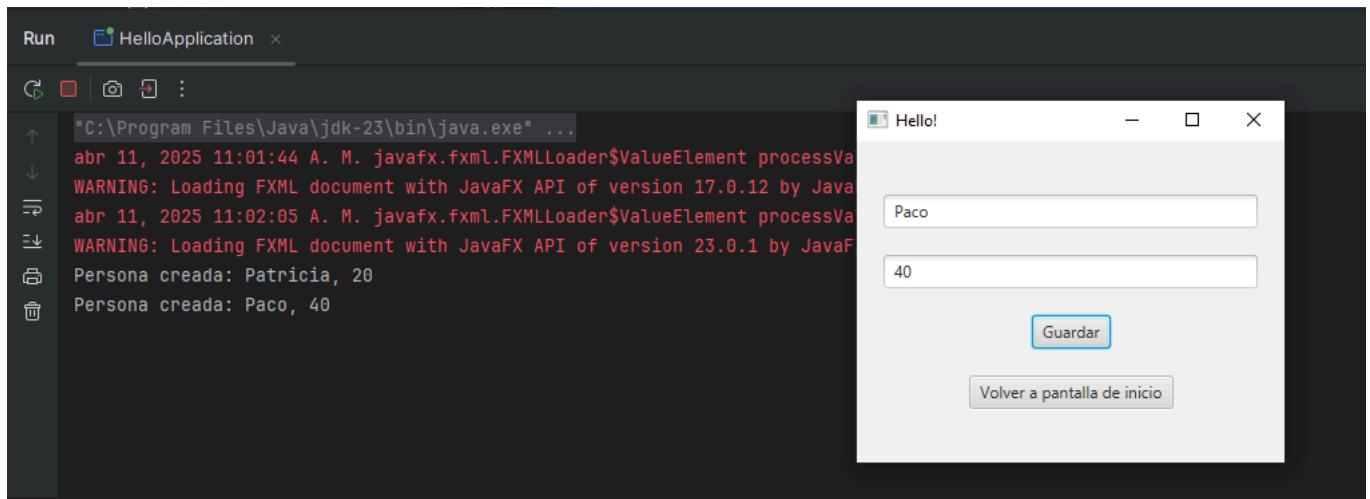
Para confirmar que hemos guardado a la persona, imprimiremos en la consola un mensaje.

Lo probamos:



A screenshot of a JavaFX application window titled "Hello!". The input field contains "Patricia" and the second input field contains "20". The "Guardar" button is highlighted with a blue border. A terminal window shows the command "C:\Program Files\Java\jdk-23\bin\java.exe" ... and log messages indicating the creation of a new person object.

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
abr 11, 2025 11:01:44 A. M. javafx.fxml.FXMLLoader$ValueElement processValueElement
WARNING: Loading FXML document with JavaFX API of version 17.0.12 by Java
abr 11, 2025 11:02:05 A. M. javafx.fxml.FXMLLoader$ValueElement processValueElement
WARNING: Loading FXML document with JavaFX API of version 23.0.1 by Java
Persona creada: Patricia, 20
```

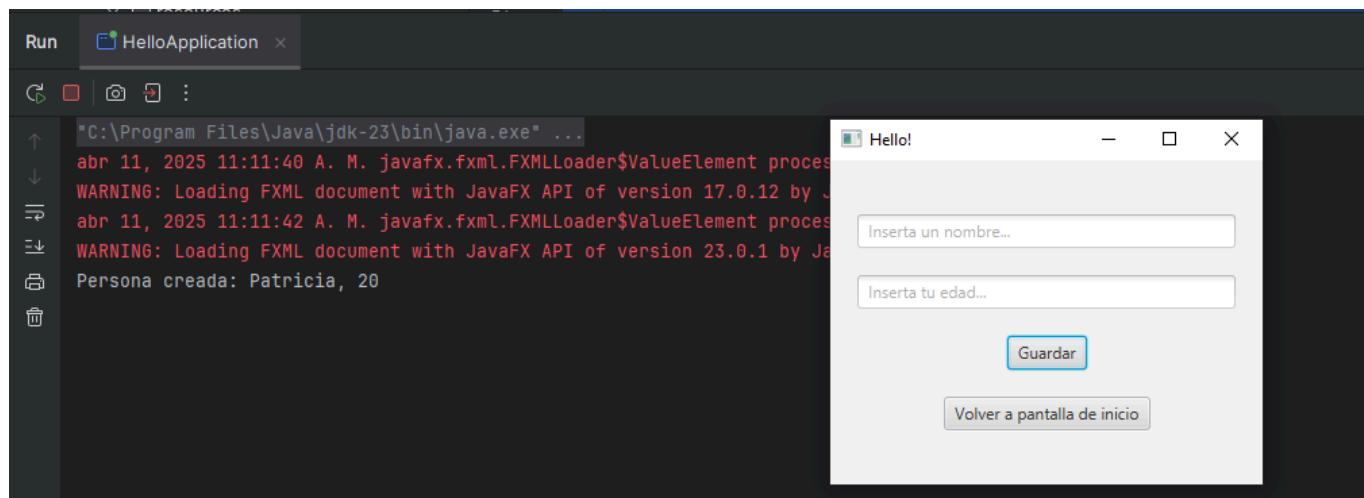


Para hacer la app un poco más curiosa, insertaremos dos `.clear()` para cada `TextField` destinados a borrar el contenido que tuvieran estos cada vez que guardamos exitosamente a la persona:

```
package org.example.demo;

import javafx.fxml.FXML;
import javafx.scene.control.TextField;
```

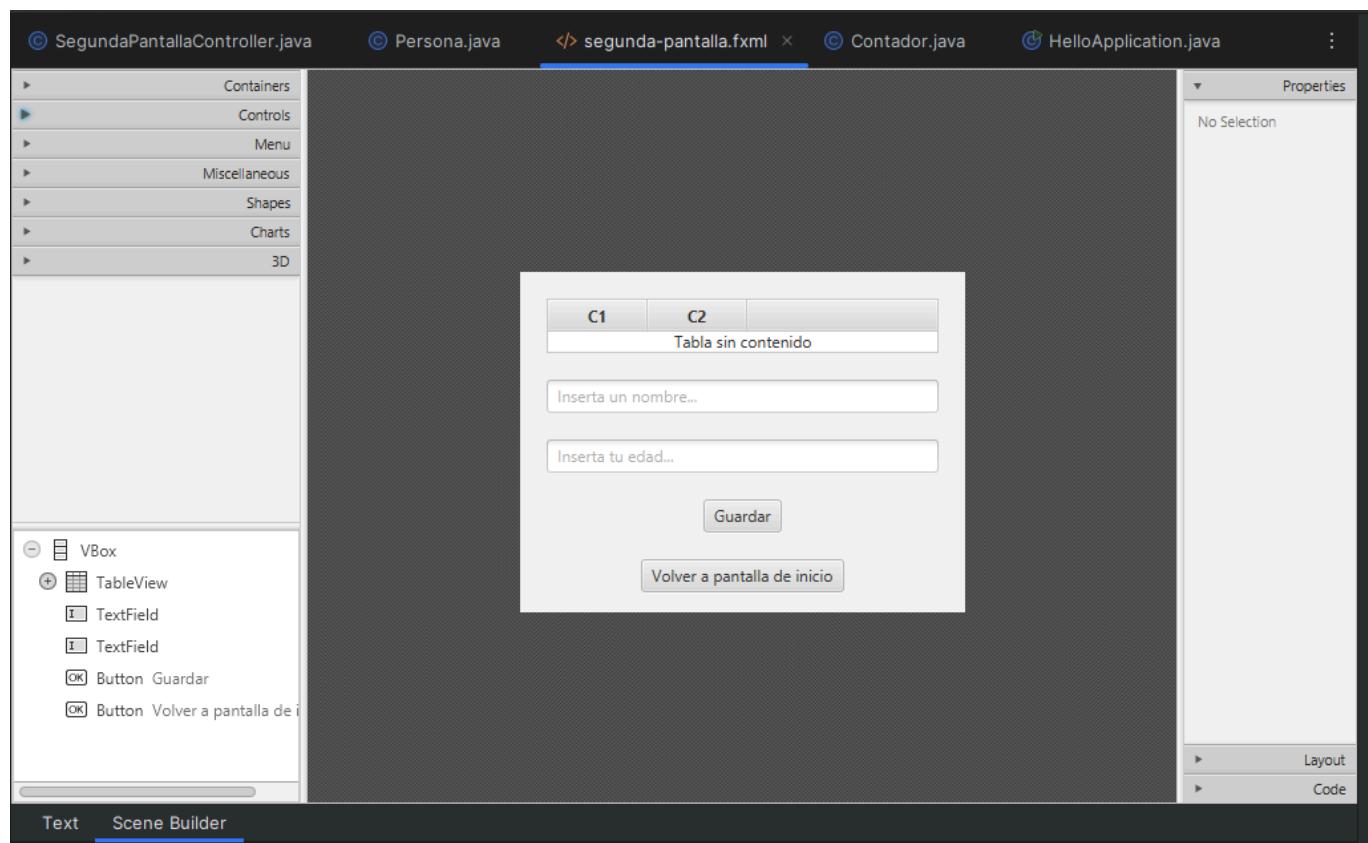
```
6 import java.io.IOException;
7
8 public class SegundaPantallaController {
9
10    @FXML
11    private TextField nombreTextField;
12
13    @FXML
14    private TextField edadTextField;
15
16    @FXML
17    private void guardarPersona() {
18
19        String nombre = nombreTextField.getText();
20        int edad;
21
22        try {
23            edad = Integer.parseInt(edadTextField.getText());
24        } catch (NumberFormatException e) {
25            System.out.println("Edad inválida");
26            return;
27        }
28
29        Persona persona = new Persona(nombre, edad);
30        System.out.println("Persona creada: " + persona.getNombre() + " "
31
32        nombreTextField.clear();
33        edadTextField.clear();
34
35    }
36
37    @FXML
38    public void irAPantallaHello() throws IOException {
39        HelloApplication.setRoot("hello-view");
40    }
41
42 }
```



8.1.6. Imprimir contenido en tablas

Para terminar con esta introducción, añadiremos a nuestra aplicación la posibilidad de imprimir en una tabla la información de las personas que vamos creando.

Insertamos una **TableView** en nuestro fxml:



y la retocamos desde el código para establecer sus propiedades:

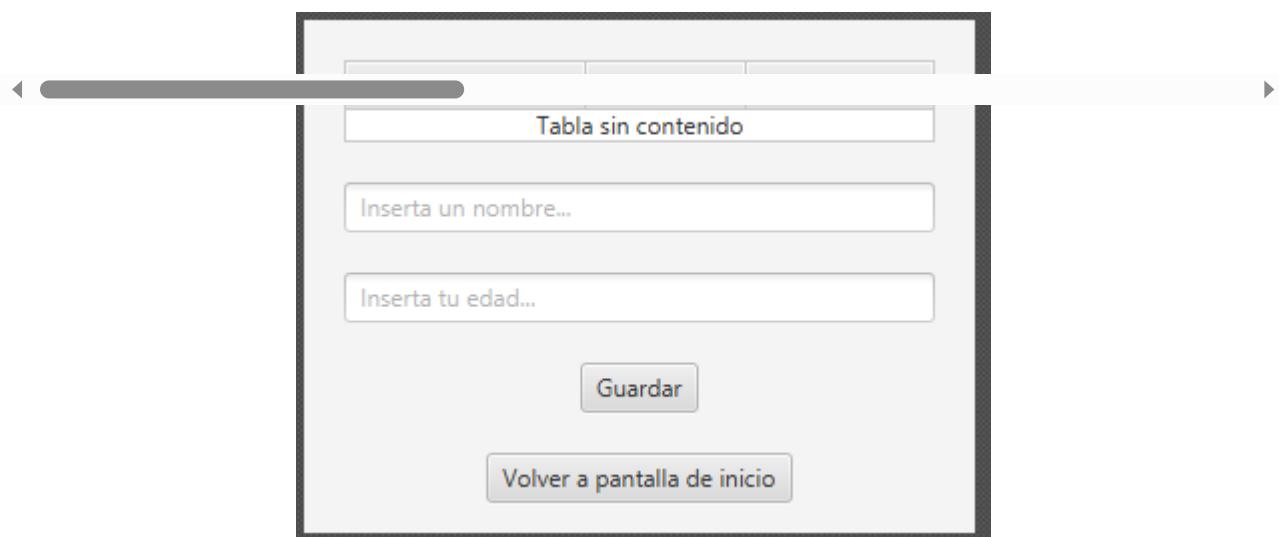
```
1 <TableView fx:id="tablaPersonas">
2     <columns>
3         <TableColumn fx:id="columnaNombre" text="Nombre" prefWidth=
4             <TableColumn fx:id="columnaEdad" text="Edad" prefWidth="80"
5         </columns>
6     </TableView>
```

Quedaría así:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

3
4 <?import javafx.geometry.*?>
5 <?import javafx.scene.control.*?>
6 <?import javafx.scene.layout.*?>
7
8 <VBox alignment="CENTER" prefHeight="247.0" prefWidth="335.0" spacing="".
9 <padding>
10 <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
11 </padding>
12 <TableView fx:id="tablaPersonas">
13   <columns>
14     <TableColumn fx:id="columnaNombre" text="Nombre" prefWidth=
15       <TableColumn fx:id="columnaEdad" text="Edad" prefWidth="80"
16     </columns>
17   </TableView>
18   <TextField fx:id="nombreTextField" promptText="Inserta un nombre...">
19   <TextField fx:id="edadTextField" promptText="Inserta tu edad..." />
20     <Button onAction="#guardarPersona" text="Guardar" />
21     <Button onAction="#irAPantallaHello" text="Volver a pantalla de ini
</VBox>
```



Ahora, desde el controlador usaremos una lista de tipo *ObservableList<Persona>*. No te preocupes, porque lo único que asuta es el nombre: el comportamiento es el mismo que el de una lista tradicional. Enlazaremos las columnas a las propiedades de *Persona*, y añadiremos *.add()* a la lista por cada nueva persona cuando se hace clic en "*Guardar*".

```

package org.example.demo;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
```

```

import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;

import java.io.IOException;

public class SegundaPantallaController {

    @FXML
    private TextField nombreTextField;

    @FXML
    private TextField edadTextField;

    @FXML
    private TableView<Persona> tablaPersonas;

    @FXML
    private TableColumn<Persona, String> columnaNombre;

    @FXML
    private TableColumn<Persona, Integer> columnaEdad;

    private ObservableList<Persona> listaPersonas = FXCollections.observableArrayList();

    @FXML
    public void initialize() {
        // enlazamos columnas con propiedades del objeto Persona
        columnaNombre.setCellValueFactory(data -> new javafx.beans.property.SimpleStringProperty(data.getValue().getNombre()));
        columnaEdad.setCellValueFactory(data -> new javafx.beans.property.SimpleIntegerProperty(data.getValue().getEdad()));

        // asignamos la lista al TableView
        tablaPersonas.setItems(listaPersonas);
    }

    @FXML
    private void guardarPersona() {

        String nombre = nombreTextField.getText();
        int edad;

        try {
            edad = Integer.parseInt(edadTextField.getText());
        } catch (NumberFormatException e) {
            System.out.println("Edad inválida");
            return;
        }
}

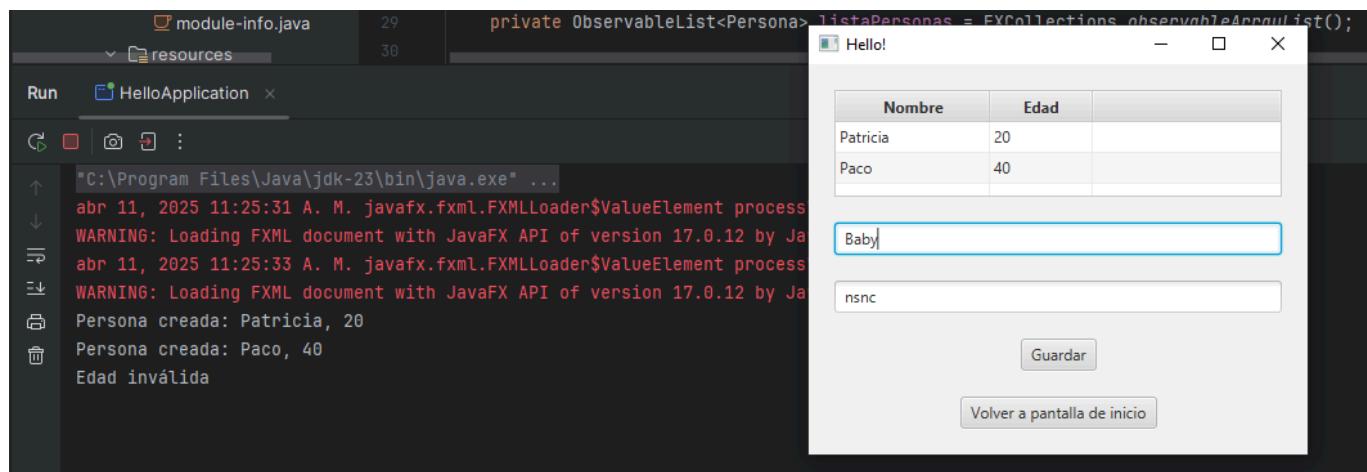
```

```

53     }
54
55     Persona persona = new Persona(nombre, edad);
56     listaPersonas.add(persona);
57     System.out.println("Persona creada: " + persona.getNombre() + "
58
59     nombreTextField.clear();
60     edadTextField.clear();
61
62 }
63
64 @FXML
65 public void irAPantallaHello() throws IOException {
66     HelloApplication.setRoot("hello-view");
67 }
68
}

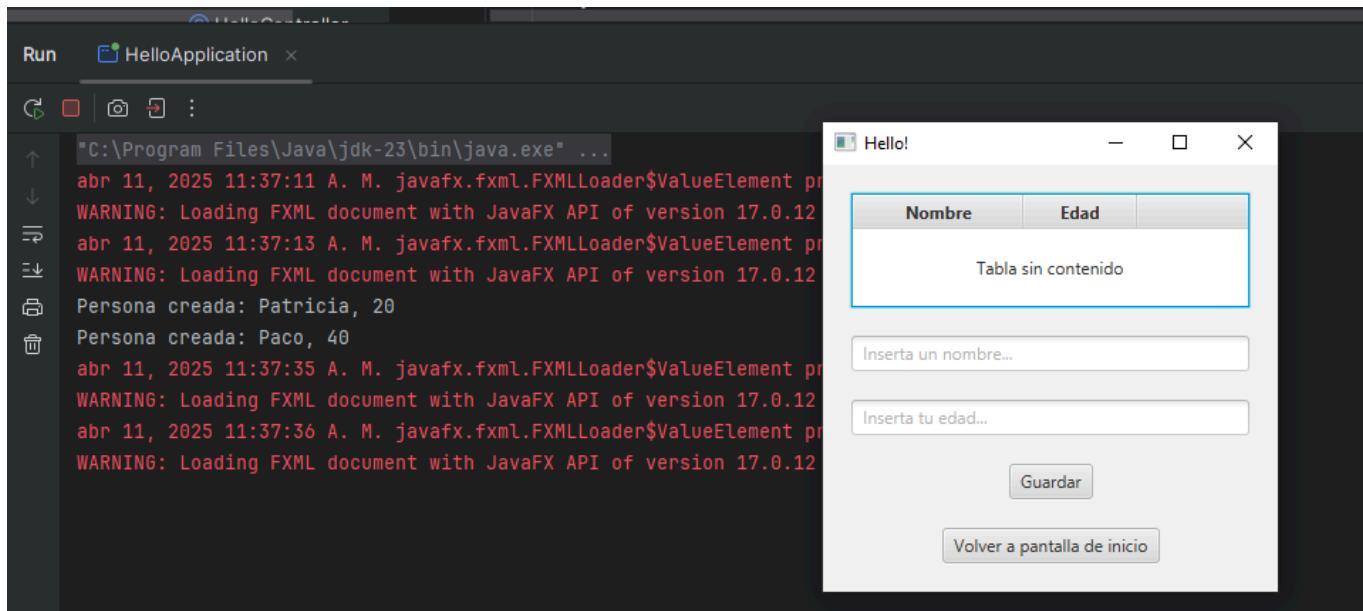
```

Lo probamos:



Haz esta prueba...

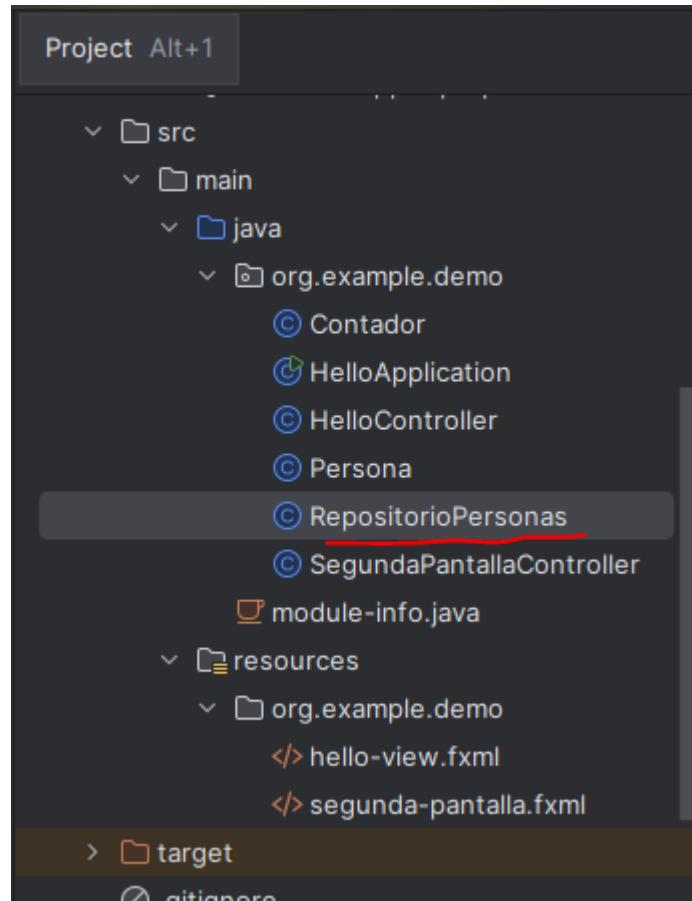
Ejecuta tu aplicación, crea a varias personas, vuelve a la pantalla principal y vuelve a entrar en tu segunda pantalla... han desaparecido las personas que teníamos en la tabla:



Eso pasa porque cuando volvemos a la pantalla de inicio, el controlador se vuelve a crear desde cero, lo que significa que se crea una nueva lista vacía (`listaPersonas = FXCollections.observableArrayList()`) y, por tanto, la tabla aparece vacía otra vez.

Para solucionarlo, usaremos una clase estática para guardar los datos de la lista:

```
1 import javafx.collections.FXCollections;
2 import javafx.collections.ObservableList;
3
4 public class RepositorioPersonas {
5     private static final ObservableList<Persona> personas = FXCollectio
6
7     public static ObservableList<Persona> getPersonas() {
8         return personas;
9     }
10
11     public static void insertarPersona(Persona persona) {
12         personas.add(persona);
13     }
14 }
```



En el `initialize()` del controlador, deberemos modificar la llamada a `.setItems()` que teníamos para adaptarlo a la nueva clase:

```
1  @FXML
2  public void initialize() {
3
4      columnaNombre.setCellValueFactory(data -> new javafx.beans.property.SimpleStringProperty());
5      columnaEdad.setCellValueFactory(data -> new javafx.beans.property.SimpleIntegerProperty());
6
7      // en lugar de crear lista propia, usamos la lista estática
8      tablaPersonas.setItems(RepositorioPersonas.getPersonas());
9
10 }
```

Y al guardar, en lugar de hacer un `.add` directamente, usaremos el método `insertarPersona()` de la nueva clase estática:

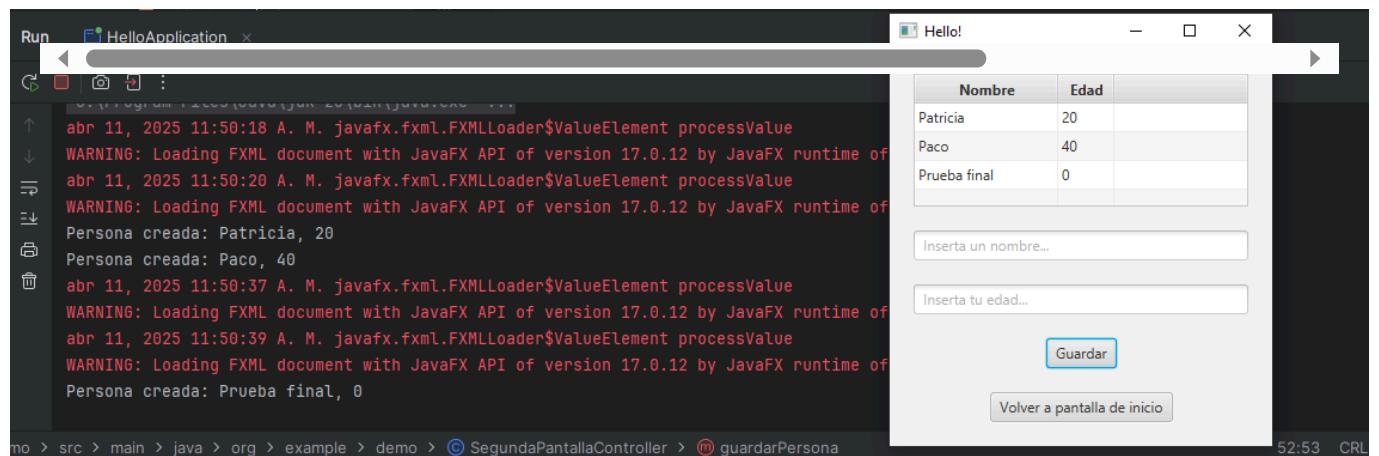
```
@FXML
private void guardarPersona() {
    String nombre = nombreTextField.getText();
```

```

6     int edad;
7
8     try {
9         edad = Integer.parseInt(edadTextField.getText());
10    } catch (NumberFormatException e) {
11        System.out.println("Edad inválida");
12        return;
13    }
14
15    Persona persona = new Persona(nombre, edad);
16
17    RepositorioPersonas.insertarPersona(persona);
18
19    System.out.println("Persona creada: " + persona.getNombre() + " "
20
21    nombreTextField.clear();
22    edadTextField.clear();
23}

```

Vuelve a probar y comprueba que ahora la lista se guarda correctamente aunque cambiemos la pantalla...



Práctica 1. Entrega tutorial: App pruebas en JavaFX

Si has seguido los pasos anteriores, crea un informe PDF sobre el proceso y súbelo a la entrega de AULES disponible.

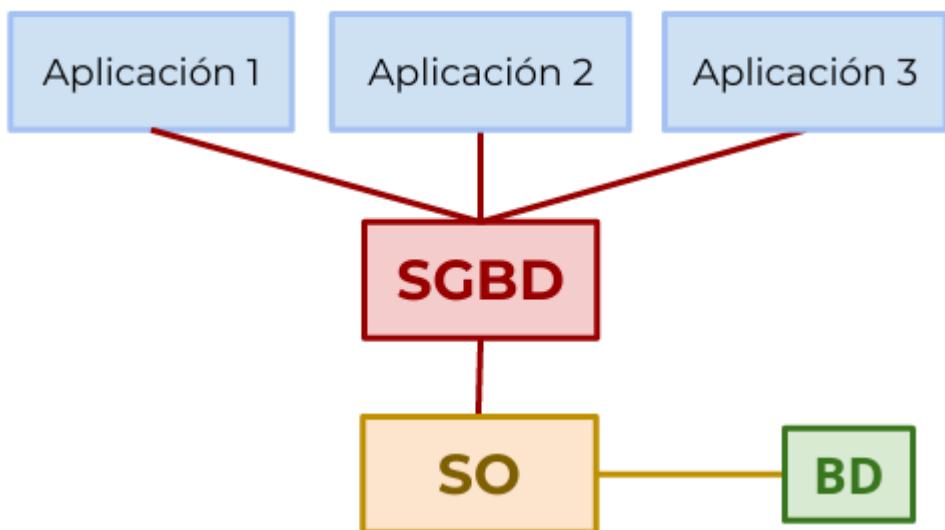


Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<<http://creativecommons.org/licenses/by-sa/4.0/>>

8_2_bbdd_relacionales

8.2. Bases de datos relacionales. MySQL/MariaDB.

Una **Base de Datos (BD)** es un conjunto de datos relacionados, que se encuentran agrupados o estructurados. Por otro lado tenemos los **Sistemas Gestores de Bases de Datos (SGBD)**, que proporcionan una interfaz entre los datos y los programas de aplicación que acceden a ellos. Se usan para guardar, organizar y acceder a estos datos de forma rápida y segura.



Ejemplos de *SGBD*:

- *MySQL*
- *MariaDB*
- *PostgreSQL*
- *Oracle Database*
- *SQL Server*

MySQL fue creado en los años 90 y es uno de los gestores de bases de datos más usados en el mundo, sobre todo para aplicaciones web. Sobre este tipo de *SGBD* corre, por ejemplo, cualquier sitio creado en *Wordpress*.

MariaDB es una versión mejorada de *MySQL* (literalmente, es un fork del [repositorio de GitHub de MySQL](https://github.com/mysql/mysql-server) <<https://github.com/mysql/mysql-server>>). Fue creada por los mismos desarrolladores originales de *MySQL* cuando Oracle compró *MySQL*. Es 100% compatible con *MySQL* y

además es gratuita y de código abierto. Ambos funcionan usando el lenguaje SQL (*Structured Query Language*).



MariaDB Foundation - MariaDB.org <<https://mariadb.org/>>

8.2.1. Instalación de XAMPP

XAMPP es un paquete de software que incluye todo lo necesario para crear un servidor web local en nuestro equipo.



XAMPP incluye:

- *Apache* → el servidor web.
- *MySQL o MariaDB* → el SGBD.
- *PHP* → lenguaje de programación del lado del servidor.
- *phpMyAdmin* → una aplicación web para gestionar bases de datos fácilmente desde el navegador.

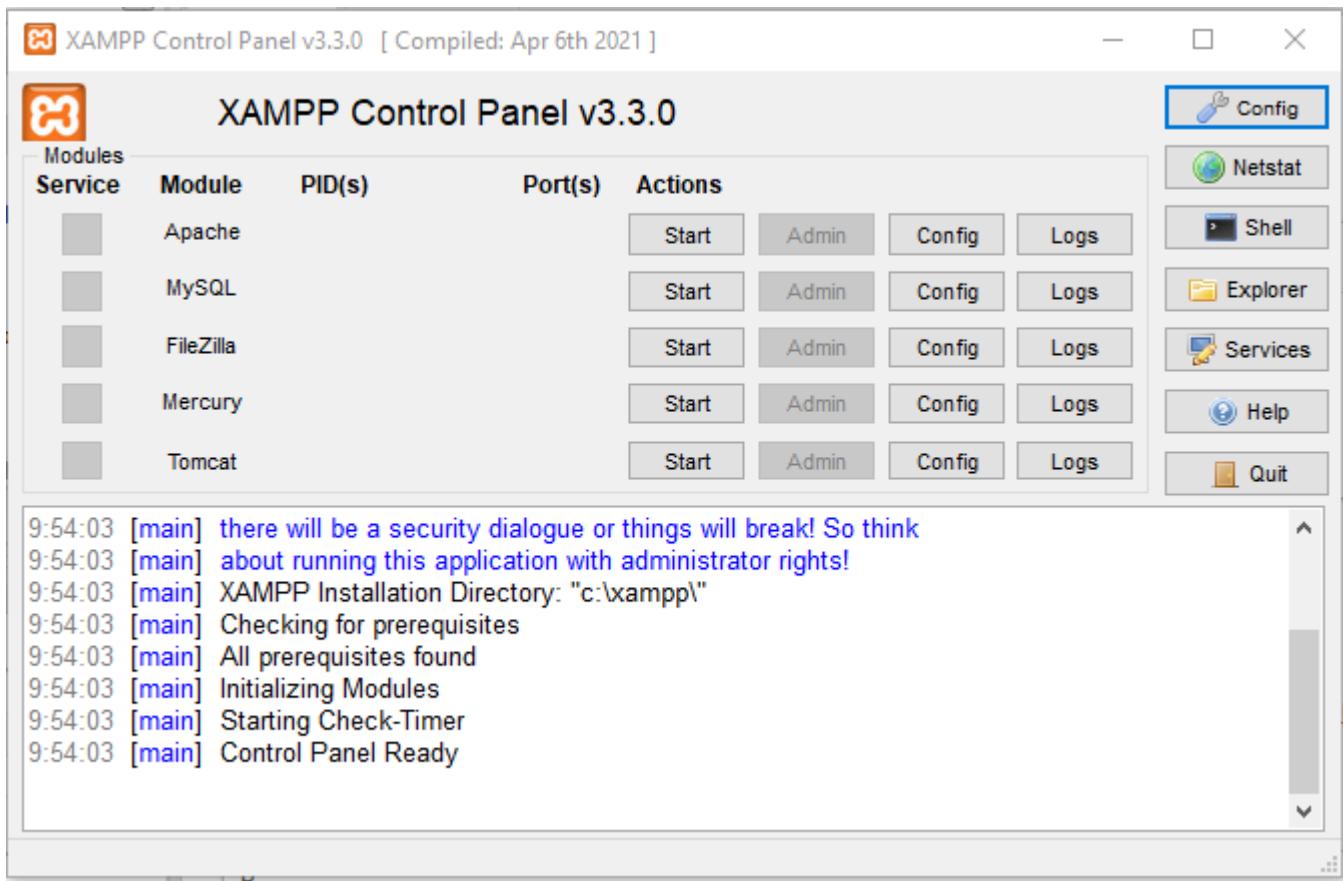
Vale, pero... ¿qué tiene que ver XAMPP?

Haremos uso de esta herramienta para gestionar nuestras bases de datos con *MySQL/MariaDB*, ya que, además de proporcionarnos este servicio, también tendremos disponible **Apache** para mejoras futuras en cuanto a servidor para implementar otro tipo de apps web (como las de IA con *TensorFlow* y *JS*).

Otra alternativa es instalar directamente el servicio de *MariaDB* **descargándolo desde su sitio web.** <<https://mariadb.org/download/>>

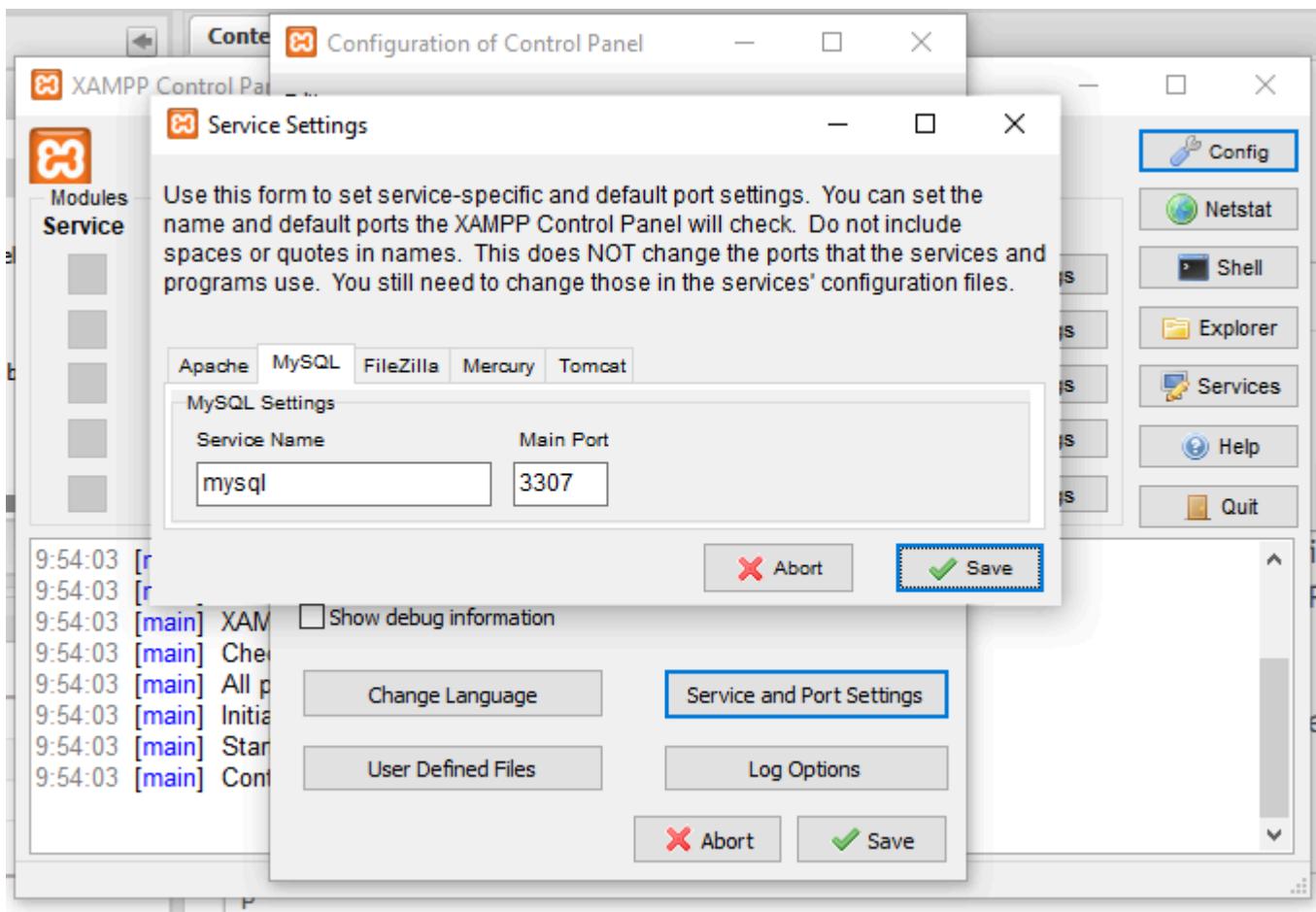
Instalación y configuración de XAMPP

Descarga e instala: **XAMPP Installers and Downloads for Apache Friends**
<<https://www.apachefriends.org/es/index.html>>

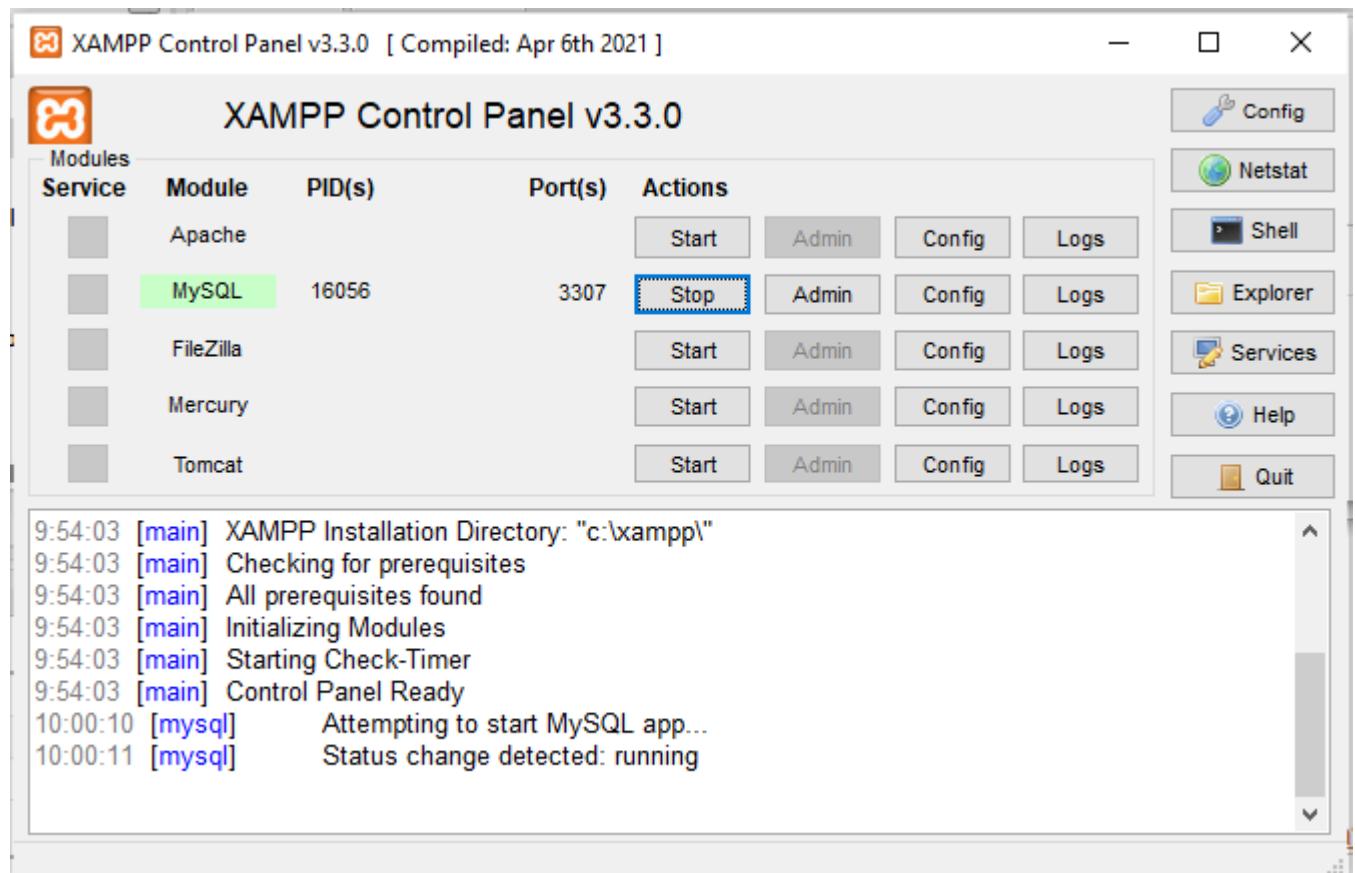


Una vez dentro, observa que aparecen los servicios y sus respectivos puertos. Por defecto, **Apache usa el puerto 80, y MySQL/MariaDB el 3306** (el mismo que usaría el SGBD sin XAMPP). Para evitar problemas por si se decide instalar un servicio de *MySQL/MariaDB* aislado de XAMPP en el futuro, **vamos a modificar el puerto por el 3307.**

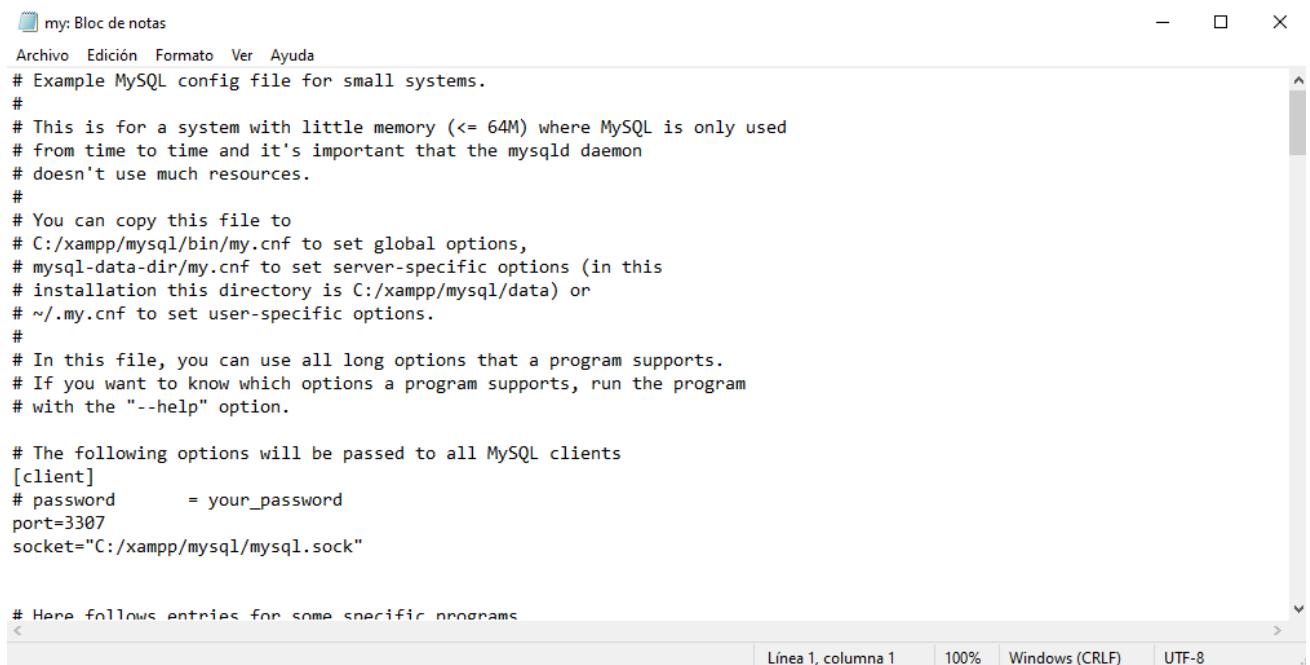
Para ello, ejecutamos **XAMPP como administrador** y nos dirigimos a *Config --> Serve and Port Settings*. En la pestaña de *MySQL* modificamos el puerto:



Inicia el servicio y comprueba que se conecta al puerto correcto (3307):



Podremos consultar el archivo de configuración principal del servicio *my.ini* desde el botón **Config**.

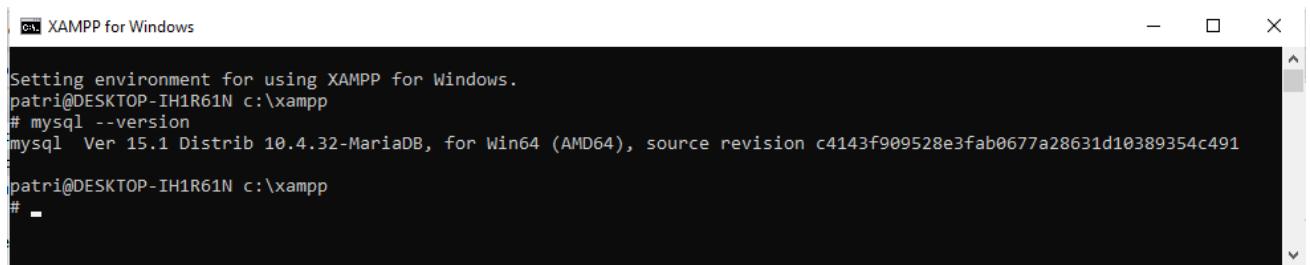


```
# Example MySQL config file for small systems.
#
# This is for a system with little memory (<= 64M) where MySQL is only used
# from time to time and it's important that the mysqld daemon
# doesn't use much resources.
#
# You can copy this file to
# C:/xampp/mysql/bin/my.cnf to set global options,
# mysql-data-dir/my.cnf to set server-specific options (in this
# installation this directory is C:/xampp/mysql/data) or
# ~/.my.cnf to set user-specific options.
#
# In this file, you can use all long options that a program supports.
# If you want to know which options a program supports, run the program
# with the "--help" option.

# The following options will be passed to all MySQL clients
[client]
password      = your_password
port=3307
socket="C:/xampp/mysql/mysql.sock"

# Here follows entries for some specific programs
```

Por último, consultamos a través del *Shell* la versión instalada de *MySQL* para comprobar que ya estamos actualizados con *MariaDB*:



```
Setting environment for using XAMPP for Windows.
patri@DESKTOP-IH1R61N c:\xampp
# mysql --version
mysql Ver 15.1 Distrib 10.4.32-MariaDB, for Win64 (AMD64), source revision c4143f909528e3fab0677a28631d10389354c491
patri@DESKTOP-IH1R61N c:\xampp
# -
```

Conexión con el servidor de *MariaDB*

Antes de irnos a un software externo, probaremos el acceso desde el mismo XAMPP. Se nos habrá creado un usuario *root* con contraseña vacía.

Desde el mismo *Shell*, tecleamos el siguiente comando para acceder a *MariaDB* por consola:

```
1 | mysql -u root -p
```

Con el parámetro *-u* le indicamos el usuario y con el parámetro *-p* la contraseña. Al no indicarle esta última, *MySQL* nos pedirá que la introduzcamos, pero como el servidor no tiene contraseña, deberemos pulsar *enter* directamente.

```
ɔ XAMPP for Windows - mysql -u root -p
patri@DESKTOP-IH1R61N c:\xampp
# mysql --version
mysql Ver 15.1 Distrib 10.4.32-MariaDB, for Win64 (AMD64), source revision c4143f909528e3fab0677a28631d10389354c491
patri@DESKTOP-IH1R61N c:\xampp
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> -
```

Una vez dentro, ya podremos consultar tablas y realizar cualquier operación con SQL de las que ya conocemos. Por hacer algo, crearemos una nueva base de datos llamada **prueba** y consultaremos las BBDD que trae por defecto *MariaDB* para configuración:

```
1 | create database prueba;
2 | show databases;
```

```
ɔ XAMPP for Windows - mysql -u root -p
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| phpmyadmin     |
| prueba         |
| test           |
+-----+
6 rows in set (0.079 sec)

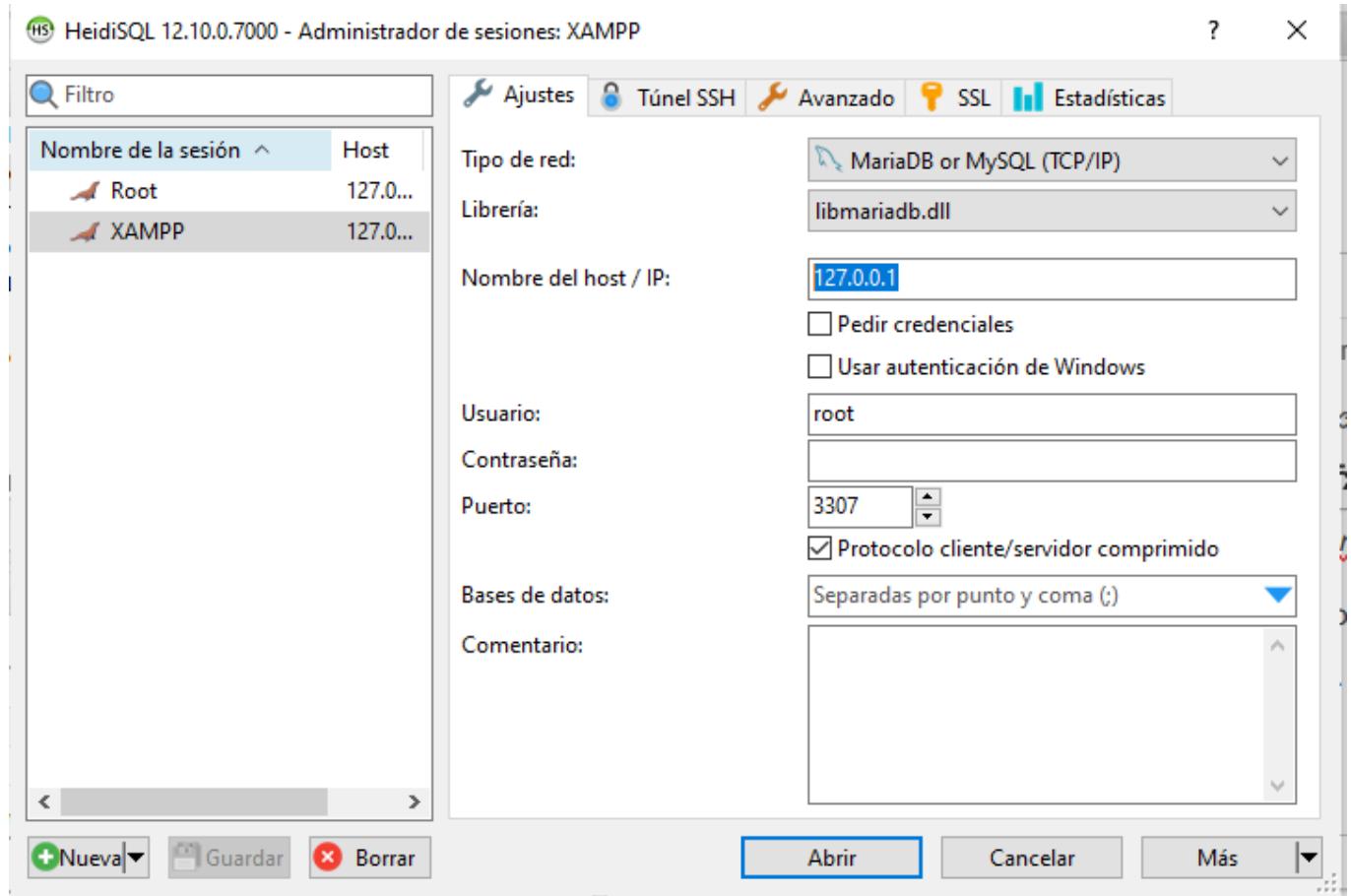
MariaDB [(none)]> -
```

Conexión desde software GUI externo

Una vez comprobado que el servicio funciona, vamos a conectarnos desde algún software externo para hacer más amigable la interacción con nuestra base de datos. Elige el que quieras o el que estés acostumbrado (*HeidiSQL*, *sqlDeveloper*, *SQLWorkbench*, etc.).

Como el nativo de *MariaDB* es **HeidiSQL** <<https://www.heidisql.com/>>, para los ejemplos usaremos ese.

Creamos una nueva conexión a nuestro servidor de *MariaDB* desde *HeidiSQL*. Por defecto, el nombre del host es siempre ***localhost*** (o la ip **127.0.0.1**):



Si todo va bien, nos cargará el entorno de nuestra conexión y las bases de datos consultadas anteriormente a través de la consola aparecerán a la izquierda de la pantalla:

```

8 /* Conectado. ID de Hilo: 9 */
9 /* Reading function definitions from C:\Program Files (x86)\Common Files\MariaDBShared\HeidiSQL\functions-mariadb.ini */
10 SHOW TABLES FROM `information_schema`;
11 SHOW DATABASES;
12 SHOW OPEN TABLES FROM prueba WHERE `in_use`!=0;
13 USE `prueba`;
14 /* Entrando a la sesión "XAMPP" */
15 SELECT `DEFAULT_COLLATION_NAME` FROM `information_schema`.`SCHEMATA` WHERE `SCHEMA_NAME`='prueba';
16 SHOW TABLE STATUS FROM prueba;
17 SHOW FUNCTION STATUS WHERE `Db`='prueba';
18 SHOW PROCEDURE STATUS WHERE `Db`='prueba';
19 SHOW TRIGGERS FROM prueba;
20 SELECT *, `EVENT_SCHEMA` AS `Db`, `EVENT_NAME` AS `Name` FROM information_schema.`EVENTS` WHERE `EVENT_SCHEMA`='prueba';
21 /* Cargando archivo "C:\Users\patr1\Documents\mundial.sql" (97 B) en pestaña de consulta #1 */
22 /* Escalando controles a DPI de pantalla: 100% */

```

8.2.2. Configuración del entorno (conector JDBC) y prueba de conexión desde Java

Para conectar un proyecto *Java* con *MySQL/MariaDB*, deberemos añadir al archivo *pom.xml* la dependencia para que se infiera el **JDBC** <<https://mariadb.com/kb/en/mariadb-connector-j/>> .

```
1 |     <dependency>
2 |         <groupId>org.mariadb.jdbc</groupId>
3 |         <artifactId>mariadb-java-client</artifactId>
4 |         <version>3.3.2</version>
5 |     </dependency>
```

Crearemos un nuevo proyecto *Java Maven* y se la añadimos:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
    modelVersion="4.0.0">

    <groupId>org.example</groupId>
    <artifactId>bases_datos_mariadb</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>23</maven.compiler.source>
        <maven.compiler.target>23</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.mariadb.jdbc</groupId>
            <artifactId>mariadb-java-client</artifactId>
            <version>3.3.2</version>
        </dependency>

    </dependencies>
```

```
</project>
```

--> Recuerda actualizar las dependencias *Maven* para que vuelva a cargar todos los cambios desde el *pom.xml*.

Para probar que funciona, vamos a la clase *Main* y creamos un nuevo método de tipo **Connection** para conectarnos a la BD llamado **conexion()**:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Connection bd = conexion();  
4     }  
5  
6     public static Connection conexion() {  
7         Connection conexion;  
8         String host = "jdbc:mariadb://localhost:3307/";  
9         String user = "root";  
10        String psw = "";  
11        String bd = "prueba";  
12        System.out.println("Conectando...");  
13  
14        try {  
15            conexion = DriverManager.getConnection(host+bd,user,psw);  
16            System.out.println("Conexión realizada con éxito.");  
17        } catch (SQLException e) {  
18            System.out.println(e.getMessage());  
19            throw new RuntimeException(e);  
20        }  
21  
22        return conexion;  
23    }  
24}  
25}  
26}  
27}
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Conectando...
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Conexión realizada con éxito.

Process finished with exit code 0
```

Esto que ves:

SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation

no es un error crítico. Es sólo una advertencia, porque el conector *MariaDB* puede usar *SLF4J* para logs si queremos, pero si no lo instalamos, simplemente no loguea mensajes. No afecta a la conexión ni al funcionamiento de la aplicación.

Cerrar conexión

No debemos dejar conexiones abiertas, por lo que es una buena práctica tener otro método para cerrar la conexión cuando finalizamos el programa. Vamos a declarar otro método *desconectar()*:

```
1  public class Main {
2      public static void main(String[] args) {
3
4          Connection bd = conexion();
5          System.out.println("Realizando consultas...");
6          desconectar(bd);
7
8      }
9
10     public static Connection conexion() {
11         Connection conexion;
12         String host = "jdbc:mariadb://localhost:3307/";
13         String user = "root";
14         String psw = "";
15         String bd = "prueba";
16         System.out.println("Conectando...");
17
18         try {
```

```

19         conexion = DriverManager.getConnection(host+bd,user,psw);
20         System.out.println("Conexión realizada con éxito.");
21     } catch (SQLException e) {
22         System.out.println(e.getMessage());
23         throw new RuntimeException(e);
24     }
25
26     return conexion;
27 }
28
29 public static void desconectar(Connection conexion){
30
31     System.out.println("Desconectando...");
32
33     try {
34         conexion.close();
35         System.out.println("Conexión finalizada.");
36     } catch (SQLException e) {
37         System.out.println(e.getMessage());
38         throw new RuntimeException(e);
39     }
40 }
41
42 }

```

```

"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Conectando...
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Conexión realizada con éxito.
Realizando consultas...
Desconectando...
Conexión finalizada.

Process finished with exit code 0

```

8.2.3. Primeras consultas SQL (select)

Crear datos

Para poder empezar a modificar datos de nuestra BD, vamos a crear una nueva tabla SQL llamada **Estudiantes**, con el siguiente formato:

```
1 | CREATE TABLE estudiantes (
2 |     nia INTEGER,
3 |     nombre VARCHAR(20),
4 |     fecha_nacimiento DATE,
5 |     CONSTRAINT pk_estudiante PRIMARY KEY (nia));
```

Lanzaremos un script para insertar datos, por ejemplo:

```
1 | INSERT INTO `estudiantes` (`nia`, `nombre`, `fecha_nacimiento`) VALUES
2 | INSERT INTO `estudiantes` (`nia`, `nombre`, `fecha_nacimiento`) VALUES
3 | INSERT INTO `estudiantes` (`nia`, `nombre`, `fecha_nacimiento`) VALUES
4 | INSERT INTO `estudiantes` (`nia`, `nombre`, `fecha_nacimiento`) VALUES
```

```
2 | CREATE DATABASE prueba;
3 |
4 | CREATE TABLE estudiantes (
5 |     nia INTEGER,
6 |     nombre VARCHAR(20),
7 |     fecha_nacimiento DATE,
8 |     CONSTRAINT pk_estudiante PRIMARY KEY (nia));
9 |
10| INSERT INTO `estudiantes` (`nia`, `nombre`, `fecha_nacimiento`) VALUES (12345678, 'Pedro', '2010-04-28');
11| INSERT INTO `estudiantes` (`nia`, `nombre`, `fecha_nacimiento`) VALUES (87654321, 'Aitana', '2005-04-10');
12| INSERT INTO `estudiantes` (`nia`, `nombre`, `fecha_nacimiento`) VALUES (123487654, 'Pepe', '2005-03-28');
13| INSERT INTO `estudiantes` (`nia`, `nombre`, `fecha_nacimiento`) VALUES (43215678, 'Carla', '2000-12-28');
14|
15| SELECT * from estudiantes;
```

#	nia	nombre	fecha_nacimiento
1	12.345.678	Pedro	2025-04-28
2	43.215.678	Carla	2000-12-28
3	87.654.321	Aitana	2005-04-10
4	123.487.654	Pepe	2005-03-28

Consultar de los datos generados desde Java

Para realizar la misma query de la línea anterior (`select * from`), es decir, consultar todos los datos de la tabla, nos crearemos un método nuevo llamado `consulta()`:

```
1 public static void consulta (Connection conexion){  
2  
3     String query = "SELECT * FROM estudiantes";  
4  
5     //necesitamos dos variables de tipo Statement y ResultSet para  
6     Statement stmt;  
7     ResultSet respuesta;  
8  
9     try {  
10         stmt = conexion.createStatement();  
11         respuesta = stmt.executeQuery(query);  
12  
13         while (respuesta.next()){//recorremos todas las filas existentes  
14             int nia = respuesta.getInt("nia");  
15             String nombre = respuesta.getString("nombre");  
16             Date fecha_nacimiento = respuesta.getDate("fecha_nacimiento");  
17             System.out.println("NIA: " + nia + " - Nombre: " + nombre);  
18         }  
19  
20     } catch (SQLException e) {  
21         System.out.println(e.getMessage());  
22         throw new RuntimeException(e);  
23     }  
24 }
```

Actualizamos el `main` para añadir la consulta y lanzamos el programa:

```
1 public static void main(String[] args) {  
2  
3     Connection bd = conexion();  
4     System.out.println("Realizando consultas...");  
5     consulta(bd);  
6     desconectar(bd);  
7  
8 }
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Conectando...
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Conexión realizada con éxito.
Realizando consultas...
NIA: 12345678 - Nombre: Pedro - Fecha de nacimiento: 2025-04-28
NIA: 43215678 - Nombre: Carla - Fecha de nacimiento: 2000-12-28
NIA: 87654321 - Nombre: Aitana - Fecha de nacimiento: 2005-04-10
NIA: 123487654 - Nombre: Pepe - Fecha de nacimiento: 2005-03-28
Desconectando...
Conexión finalizada.

Process finished with exit code 0
```

8.2.4. Acciones DML (*insert*, *update*, *delete*)

Para realizar las operaciones DML básicas de SQL (*insert*, *update* y *delete*) seguiremos necesitando una variable de tipo **Statement**, pero nos ahorraremos el resultado (**ResultSet**), ya que no nos interesa la respuesta del SGBD (solamente queremos que realice la acción). El método que usaremos para todas es `.executeUpdate(query)`.

Insertar datos (*INSERT*)

```
1 public static void insertar(Connection conexion){  
2  
3     String query = "INSERT INTO estudiantes (nia, nombre, fecha_nac  
4  
5     Statement stmt;  
6  
7     try {  
8         stmt = conexion.createStatement();  
9         stmt.executeUpdate(query);  
10    } catch (SQLException e) {  
11        System.out.println(e.getMessage());  
12        throw new RuntimeException(e);  
13    }  
14  
15}  
16 }
```

Modificamos el *main* para consultar los datos antes y después de insertar:

```
1 public static void main(String[] args) {  
2  
3     Connection bd = conexion();  
4     System.out.println("Realizando consultas...");  
5     consulta(bd);  
6     insertar(bd);  
7     consulta(bd);  
8     desconectar(bd);  
9  
10 }
```

```
Conexión realizada con éxito.  
Realizando consultas...  
NIA: 12345678 - Nombre: Pedro - Fecha de nacimiento: 2025-04-28  
NIA: 43215678 - Nombre: Carla - Fecha de nacimiento: 2000-12-28  
NIA: 87654321 - Nombre: Aitana - Fecha de nacimiento: 2005-04-10  
NIA: 123487654 - Nombre: Pepe - Fecha de nacimiento: 2005-03-28  
NIA: 12345678 - Nombre: Pedro - Fecha de nacimiento: 2025-04-28  
NIA: 43214321 - Nombre: Patricia - Fecha de nacimiento: 1900-04-19  
NIA: 43215678 - Nombre: Carla - Fecha de nacimiento: 2000-12-28  
NIA: 87654321 - Nombre: Aitana - Fecha de nacimiento: 2005-04-10  
NIA: 123487654 - Nombre: Pepe - Fecha de nacimiento: 2005-03-28  
Desconectando...  
Conexión finalizada.
```

```
Process finished with exit code 0
```

NOTA: Lo normal será pedir los valores al usuario y formar la *query* dinámicamente concatenando los valores en un *StringBuilder*.

(!) Cuidado cuando vuelvas a lanzar este método. ¡Si no modificas los datos del INSERT fallará por error de clave duplicada!

Modificar datos (*UPDATE*)

Es exactamente igual que el insert anterior, pero modificando el contenido de la *query*.

```
1 public static void modificar(Connection conexion){  
2  
3     System.out.println("Modificando...");  
4  
5     String query = "UPDATE estudiantes SET nombre = 'Patri' WHERE n  
6  
7     Statement stmt;  
8  
9     try {  
10         stmt = conexion.createStatement();  
11         stmt.executeUpdate(query);  
12     } catch (SQLException e) {  
13         System.out.println(e.getMessage());  
14         throw new RuntimeException(e);  
15     }  
16 }  
17 }
```

```
1  public static void main(String[] args) {
2      Connection bd = conexion();
3      System.out.println("Realizando consultas...");
4      consulta(bd);
5      modificar(bd);
6      consulta(bd);
7      desconectar(bd);
8  }
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
am Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.1\lib\idea_rt.jar=53000,127.0.0.1:53000" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.1\lib\idea_rt.jar=53000,127.0.0.1:53000" -Dfile.encoding=UTF-8 -classpath "C:\Users\Diego\OneDrive\Escritorio\Inteligencia Artificial\Inteligencia Artificial\src\main\java\com\diegocarmona\inteligenciaartificial\Controlador.java"
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Conexión realizada con éxito.
Realizando consultas...
NIA: 12345678 - Nombre: Pedro - Fecha de nacimiento: 2025-04-28
NIA: 43214321 - Nombre: Patricia - Fecha de nacimiento: 1900-04-19
NIA: 43215678 - Nombre: Carla - Fecha de nacimiento: 2000-12-28
NIA: 87654321 - Nombre: Aitana - Fecha de nacimiento: 2005-04-10
NIA: 123487654 - Nombre: Pepe - Fecha de nacimiento: 2005-03-28
Modificando...
NIA: 12345678 - Nombre: Pedro - Fecha de nacimiento: 2025-04-28
NIA: 43214321 - Nombre: Patri - Fecha de nacimiento: 1900-04-19
NIA: 43215678 - Nombre: Carla - Fecha de nacimiento: 2000-12-28
NIA: 87654321 - Nombre: Aitana - Fecha de nacimiento: 2005-04-10
NIA: 123487654 - Nombre: Pepe - Fecha de nacimiento: 2005-03-28
Desconectando...
Conexión finalizada.

Process finished with exit code 0
```

Borrar datos (*DELETE*)

```
public static void borrar(Connection conexion){

    System.out.println("Borrando...");

    String query = "DELETE FROM estudiantes WHERE nombre = 'Patri'"

    Statement stmt;

    try {
```

```
10         stmt = conexion.createStatement();
11         stmt.executeUpdate(query);
12     } catch (SQLException e) {
13         System.out.println(e.getMessage());
14         throw new RuntimeException(e);
15     }
16 }
17 }
```

```
1 public static void main(String[] args) {
2     Connection bd = conexion();
3     System.out.println("Realizando consultas...");
4     consulta(bd);
5     borrar(bd);
6     consulta(bd);
7     desconectar(bd);
8 }
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Conectando...
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Conexión realizada con éxito.
Realizando consultas...
NIA: 12345678 - Nombre: Pedro - Fecha de nacimiento: 2025-04-28
NIA: 43214321 - Nombre: Patri - Fecha de nacimiento: 1900-04-19
NIA: 43215678 - Nombre: Carla - Fecha de nacimiento: 2000-12-28
NIA: 87654321 - Nombre: Aitana - Fecha de nacimiento: 2005-04-10
NIA: 123487654 - Nombre: Pepe - Fecha de nacimiento: 2005-03-28
Borrando...
NIA: 12345678 - Nombre: Pedro - Fecha de nacimiento: 2025-04-28
NIA: 43215678 - Nombre: Carla - Fecha de nacimiento: 2000-12-28
NIA: 87654321 - Nombre: Aitana - Fecha de nacimiento: 2005-04-10
NIA: 123487654 - Nombre: Pepe - Fecha de nacimiento: 2005-03-28
Desconectando...
Conexión finalizada.

Process finished with exit code 0
|
```

8.2.5. Cargar datos de una tabla en una lista de objetos

Para guardar los resultados de una *select* en una lista de objetos tomaremos como referencia el método *consultar()*, ya que seguiremos el mismo patrón, pero en lugar de limitarnos a imprimir la información, la iremos almacenando en un *ArrayList* que mostraremos después de cargarlo por completo.

Primero, crearemos la clase **Estudiante**, con la siguiente estructura:

```
1 @Getter  
2 @AllArgsConstructor  
3 @ToString  
4 public class Estudiante {  
5  
6     private int nia;  
7     private String nombre;  
8     private Date fecha_nacimiento;  
9  
10 }
```

Y ahora, en nuestro programa principal, duplicaremos el método *consultar()* para adaptarlo a la lista que necesitamos:

```
public static void consulta_a_lista (Connection conexion){  
  
    String query = "SELECT * FROM estudiantes";  
  
    //necesitamos el dos variables de tipo Statement y ResultSet pa  
    Statement stmt;  
    ResultSet respuesta;  
    ArrayList<Estudiante> listaEstudiantes = new ArrayList<>();  
  
    try {  
        stmt = conexion.createStatement();  
        respuesta = stmt.executeQuery(query);  
  
        while (respuesta.next()){ //recorremos todas las filas exist  
            int nia = respuesta.getInt("nia");  
            String nombre = respuesta.getString("nombre");  
    }  
}
```

```

17         Date fecha_nacimiento = respuesta.getDate("fecha_nacimiento");
18         listaEstudiantes.add(new Estudiante(nia,nombre,fecha_nacimiento));
19     }
20
21     System.out.println(listaEstudiantes);
22
23 } catch (SQLException e) {
24     System.out.println(e.getMessage());
25     throw new RuntimeException(e);
26 }
27 }
```

El *main* quedaría:

```

1 public static void main(String[] args) {
2     Connection bd = conexion();
3     System.out.println("Realizando consultas...");
4     consulta_a_lista(bd);
5     desconectar(bd);
6 }
```

```

"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Conectando...
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Conexión realizada con éxito.
Realizando consultas...
[Bar] Desconectando...
Conexión finalizada.

Process finished with exit code 0
```

La lista solamente existe dentro del método *consulta_a_lista()*, lo cual no tiene mucho sentido, ya que podríamos necesitar modificarla para cambiar o borrar algún elemento (desde los métodos con el *update* y *delete*). Lo normal será **hacerla estática y cada vez que llamemos al método *consulta_a_lista()*, vaciarla para volver a cargarla.**

```

public class Main {

    static ArrayList<Estudiante> listaEstudiantes = new ArrayList<>();

    public static void main(String[] args) {
        Connection bd = conexion();
        System.out.println("Realizando consultas...");
        consulta_a_lista();
```

```

        desconectar(bd);
    }

    public static Connection conexion() {
        Connection conexion;
        String host = "jdbc:mariadb://localhost:3307/";
        String user = "root";
        String psw = "";
        String bd = "prueba";
        System.out.println("Conectando...");

        try {
            conexion = DriverManager.getConnection(host+bd,user,psw);
            System.out.println("Conexión realizada con éxito.");
        } catch (SQLException e) {
            System.out.println(e.getMessage());
            throw new RuntimeException(e);
        }

        return conexion;
    }

    public static void desconectar(Connection conexion){
        System.out.println("Desconectando...");

        try {
            conexion.close();
            System.out.println("Conexión finalizada.");
        } catch (SQLException e) {
            System.out.println(e.getMessage());
            throw new RuntimeException(e);
        }
    }

    public static void consulta_a_lista (Connection conexion){

        listaEstudiantes.clear();

        String query = "SELECT * FROM estudiantes";

        //necesitamos el dos variables de tipo Statement y ResultSet pa
        Statement stmt;
        ResultSet respuesta;

        try {

```

```

stmt = conexion.createStatement();
respuesta = stmt.executeQuery(query);

while (respuesta.next()){ //recorremos todas las filas existentes
    int nia = respuesta.getInt("nia");
    String nombre = respuesta.getString("nombre");
    Date fecha_nacimiento = respuesta.getDate("fecha_nacimiento");
    listaEstudiantes.add(new Estudiante(nia,nombre,fecha_nacimiento));
}

System.out.println(listaEstudiantes);

} catch (SQLException e) {
    System.out.println(e.getMessage());
    throw new RuntimeException(e);
}
}

public static void insertar(Connection conexion){

System.out.println("Insertando...");

String query = "INSERT INTO estudiantes (nia, nombre, fecha_nacimiento) VALUES (1, 'Patri', '1998-01-01')";

Statement stmt;

try {
    stmt = conexion.createStatement();
    stmt.executeUpdate(query);
} catch (SQLException e) {
    System.out.println(e.getMessage());
    throw new RuntimeException(e);
}
}

public static void modificar(Connection conexion){

System.out.println("Modificando...");

String query = "UPDATE estudiantes SET nombre = 'Patri' WHERE nia = 1";

Statement stmt;

try {

```

```

102         stmt = conexion.createStatement();
103         stmt.executeUpdate(query);
104     } catch (SQLException e) {
105         System.out.println(e.getMessage());
106         throw new RuntimeException(e);
107     }
108 }
109
110 public static void borrar(Connection conexion){
111
112     System.out.println("Borrando...");
113
114     String query = "DELETE FROM estudiantes WHERE nombre = 'Patri'";
115
116     Statement stmt;
117
118     try {
119         stmt = conexion.createStatement();
120         stmt.executeUpdate(query);
121     } catch (SQLException e) {
122         System.out.println(e.getMessage());
123         throw new RuntimeException(e);
124     }
125 }
126
127 }
128
129 }
```

Otra opción es no definir la lista estática en la clase, y simplemente que el método *consulta_a_lista()* devuelva un *ArrayList* que se cree cada vez que lo llamemos:

```

public static ArrayList<Estudiante> consulta_a_lista (Connection conexi
ArrayList<Estudiante> listaEstudiantes = new ArrayList<>();

String query = "SELECT * FROM estudiantes";

//necesitamos el dos variables de tipo Statement y ResultSet pa
Statement stmt;
ResultSet respuesta;

try {
```

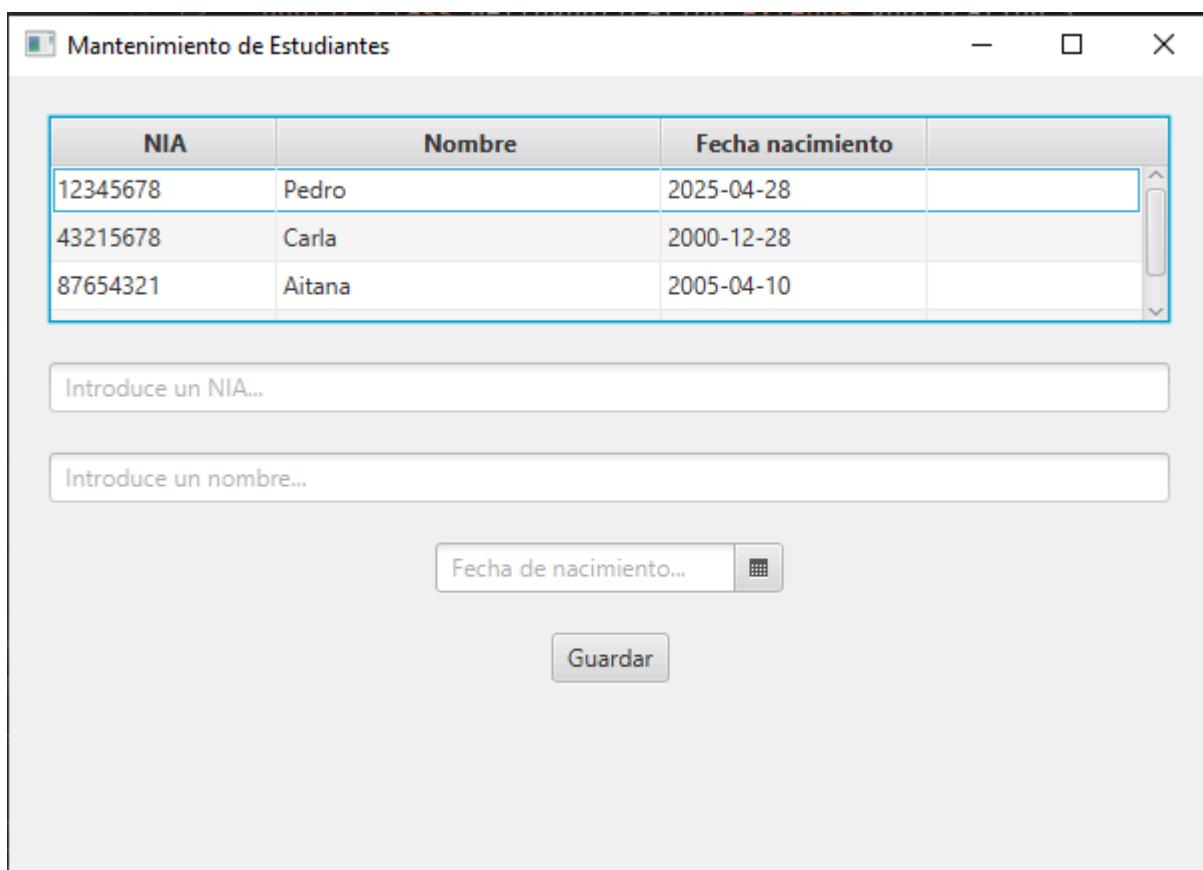
```
12         stmt = conexion.createStatement();
13         respuesta = stmt.executeQuery(query);
14
15         while (respuesta.next()){ //recorremos todas las filas existentes
16             listaEstudiantes.add(new Estudiante(respuesta.getInt("nro"),
17                                                 respuesta.getString("nombre"),
18                                                 respuesta.getDate("fecha_nacimiento")));
19         }
20
21         System.out.println(listaEstudiantes);
22
23     } catch (SQLException e) {
24         System.out.println(e.getMessage());
25         throw new RuntimeException(e);
26     }
27
28     return listaEstudiantes;
29
30 }
```

En general, nos quedaremos con esta última opción para trabajar con nuestras apps.

Práctica 2. Mi primer CRUD: mostrar y manipular datos desde una TableView de JavaFX

CRUD es el acrónimo de *Create, Read, Update and Delete* (Crear, Leer, Actualizar y Borrar), que se usa para referirse a las funciones básicas en bases de datos.

- a) Debes crear para **Estudiantes** una pantalla de mantenimiento similar a la realizada anteriormente en la Práctica 1 con JavaFX, donde almacenábamos la info escrita en dos *TextField* en una lista mediante el botón *Guardar* y la mostrábamos en una *TableView*.



La fecha de nacimiento es un elemento *FXML* de tipo *DatePicker*, y el tipo de dato *LocalDate*.

La diferencia es que ahora cuando *clickemos* en *Guardar* tendremos que almacenar a los estudiantes en una tabla de BD, y no en una lista estática (se cambia el *.add()* por un *INSERT*).

- b) Para mostrar los datos de los estudiantes guardados en una *TableView*, realizaremos una consulta (*select*) a la BD, almacenaremos los resultados en una lista *ObservableList* (como en el apartado 8.2.5.) y se la devolveremos a la tabla para que la imprima.

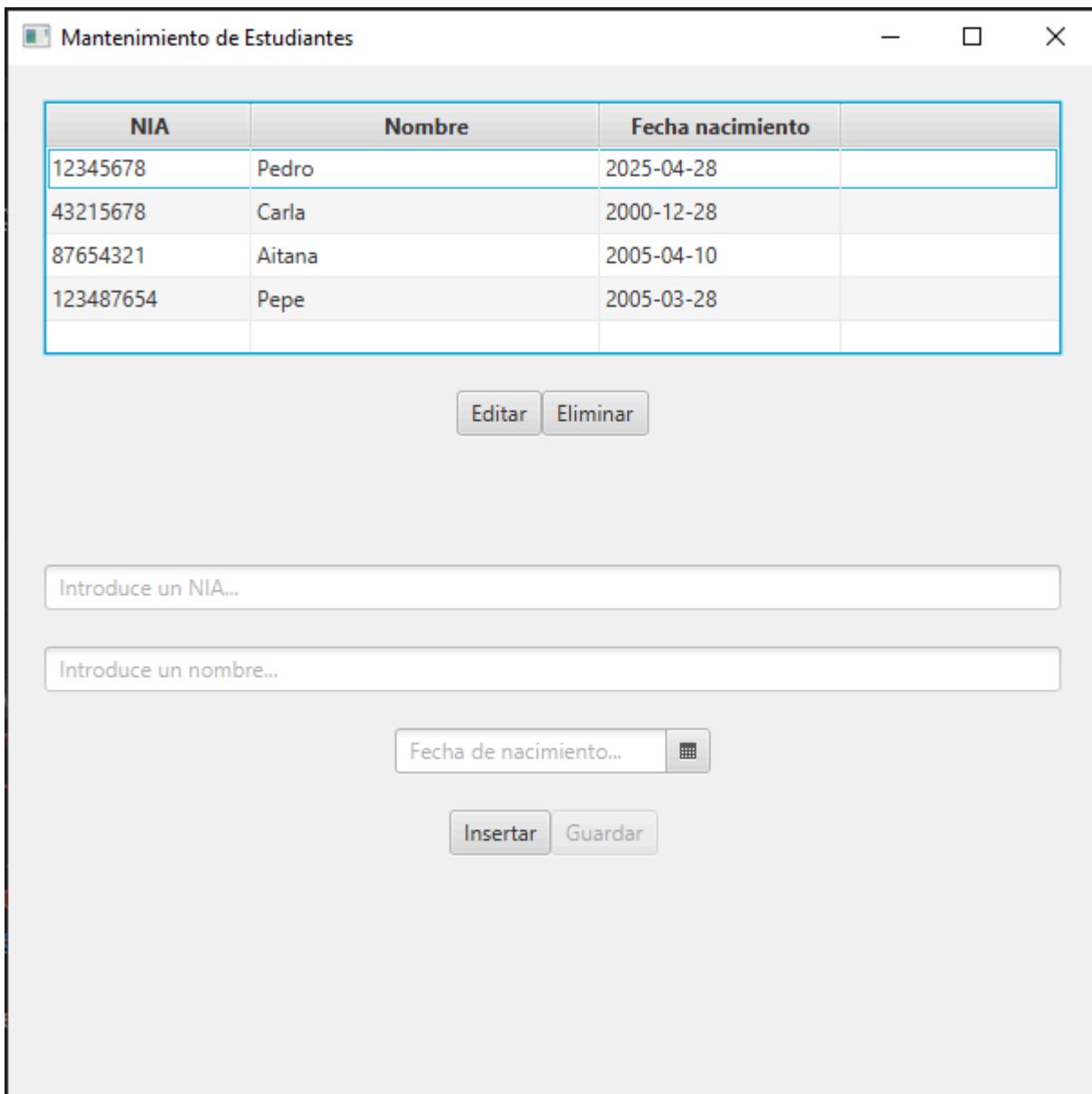
Para mostrar el formato correcto de la columna de la fecha de nacimiento debe configurarse en el método *initialize()* de la siguiente manera:

```

1 |         columnaFechaNac.setCellValueFactory(data ->
2 |             new ReadOnlyObjectWrapper<>(data.getValue().getFecha_na
3 |         );

```

- c) Cámbiale el nombre al botón **Guardar**. llámalo **Insertar**. Añade tres botones más a tu app: **Actualizar**, **Eliminar** y **Guardar** (que ahora servirá para otra cosa).



El botón **Guardar** estará inhabilitado por defecto.

- d) El botón **Eliminar** debe lanzar una instrucción a la BD de tipo DELETE, con los valores de los campos a eliminar del registro de la tabla que esté seleccionado.

Para obtener los valores de la fila seleccionada y poder construir el *WHERE* del *DELETE*, usa este código de referencia:

```
1 Estudiante seleccionado = tablaEstudiantes.getSelectionModel().  
2  
3 if (seleccionado != null) {  
4     Mantenimiento.borrar(bd,seleccionado);  
5 }else{  
6     System.out.println("No hay ninguna fila seleccionada.");  
7 }
```

- e) Además, al seleccionar una fila en la *TableView* y hacer *click* en **Editar**, se cargarán los datos visiblemente en los *TextField* y *DatePicker* para poder editarlos.

```
1 Estudiante seleccionado = tablaEstudiantes.getSelectionModel().g  
2  
3 if (seleccionado != null) {  
4     niaTextField.setText(String.valueOf(seleccionado.getNia()));  
5     nombreTextField.setText(seleccionado.getNombre());  
6     fechaNacPicker.setValue(seleccionado.getFecha_nacimiento());  
7 }else{  
8     System.out.println("No hay ninguna fila seleccionada.");  
9 }
```

Mantenimiento de Estudiantes

NIA	Nombre	Fecha nacimiento
12345678	Pedro	2025-04-28
43215678	Carla	2000-12-28
87654321	Aitana	2005-04-10
123487654	Pepe	2005-03-28

Editar **Eliminar**

43215678

Carla

28/12/2000

Insertar **Guardar**

Se habilitará el botón **Guardar** y se inhabilitará el botón **Insertar** mientras dure la edición. Usa esto para modificar la propiedad de habilitar / inhabilitar los botones:

```
1 | boton.setDisable(true);
```

f) El *click* en el botón **Guardar** hará un *UPDATE* a la tabla de la BD con los valores que tengan los *TextField* y *DatePicker* en el momento de ser presionado. Cuando finalice la edición, se debe volver a inhabilitar el botón **Guardar** y restablecer como habilitado el botón **Insertar**.

Exportación (generar .exe)

Para exportar tu aplicación JavaFX a un archivo ejecutable, como un .exe (para Windows), seguiremos los siguientes pasos.

Paso 1: Construir un archivo JAR de tu aplicación

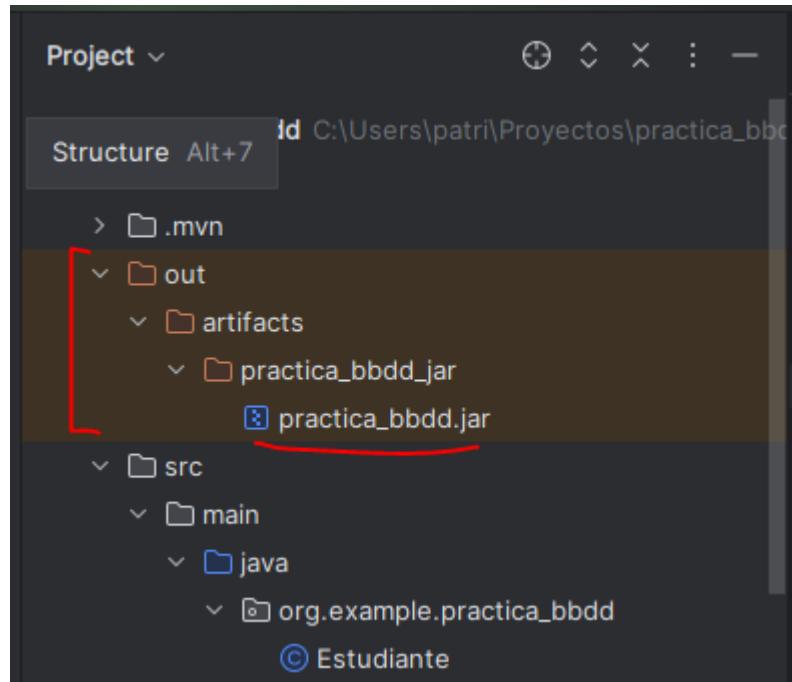
Configura el archivo .jar:

- Ve a **File > Project Structure**.
- En **Artifacts**, haz clic en el ícono de más (+) y selecciona **JAR > From modules with dependencies**.
- En la ventana que aparece, selecciona el módulo que contiene tu clase principal.
- En el campo "**Main Class**", selecciona la clase que contiene el método main (por ejemplo, org.example.Main).
- Haz clic en OK para cerrar el cuadro de configuración.

Generar el archivo .jar:

- Ve a **Build > Build Artifacts > MiApp > Build**.

IntelliJ IDEA generará el archivo .jar y lo guardará en la carpeta **out/artifacts**.



Paso 2. Usar Launch4j.

Launch4j es una herramienta que se usa para crear ejecutables .exe para aplicaciones Java.



Descárgala aquí: [Launch4j <https://launch4j.sourceforge.net/>](https://launch4j.sourceforge.net/)

Abre la aplicación y configura los campos como sigue:

- **Jar File:** Tu archivo .jar generado anteriormente.
- **Output File:** El nombre y la ruta del archivo .exe que deseas generar (por ejemplo, MiAplicacion.exe).
- **Main Class:** La clase principal de tu aplicación (por ejemplo, *org.example.Main*).
- Haz clic en **Build Wrapper** para generar el .exe.



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<http://creativecommons.org/licenses/by-sa/4.0/>

BONUS. IA con teachable machine

BONUS. Herramientas IA para la implementación de modelos de aprendizaje automático en aplicaciones (Teachable Machine + Tensorflow).

Teachable Machine es una herramienta creada por *Google* que permite a cualquier persona—sin necesidad de saber programación— entrenar modelos de *machine learning* de forma sencilla.



The screenshot shows the Teachable Machine website. At the top, there are navigation links: 'Acerca de', 'Preguntas frecuentes', and a blue button 'Primeros pasos'. Below this, the main title 'Teachable Machine' is displayed in large blue letters. A sub-section titled 'Prepara a un ordenador para que reconozca tus imágenes, sonidos y posturas.' is shown with a description of how it allows users to quickly create AI models for web sites, applications, and much more without specialized knowledge or programming. A 'Primeros pasos' button is located at the bottom left of this section. At the bottom of the page, there are icons for TensorFlow.js, p5.js, Coral, Node.js, and Arduino.

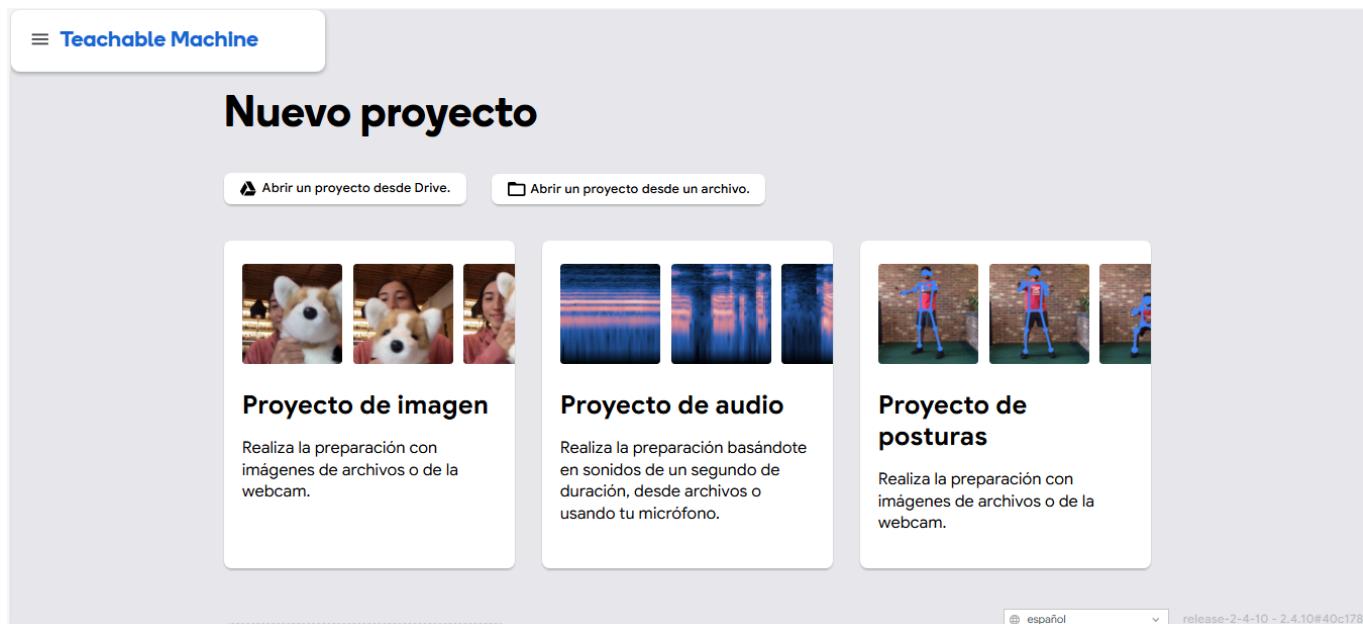
Acceso a **Teachable Machine** <<https://teachablemachine.withgoogle.com/>>

Con *Teachable Machine* podremos crear:

- Modelos de reconocimiento de imágenes, sonidos o posturas (movimientos del cuerpo).
- Entrenar el modelo subiendo tus propios datos (por ejemplo, fotos o grabaciones de audio).
- Exportarlo para usarlo en proyectos web, aplicaciones o dispositivos como *Arduino*.

Todo el entrenamiento se realiza directamente en cualquier navegador, así que es muy accesible.

Creación del modelo con Teachable Machine



≡ Teachable Machine

Class 1

Añadir muestras de imágenes:

Webcam Subir

Class 2

Añadir muestras de imágenes:

Webcam Subir

Preparación

Preparar modelo

Avanzado

Vista previa

Para obtener una vista previa de un modelo aquí, primero debes prepararlo en la parte de la izquierda.

Exportar modelo

Añadir una clase

español release-2-4-10 - 2.4.10#40c178

≡ Teachable Machine

Frente

65 muestras de imágenes

Webcam Subir

Perfil

54 muestras de imágenes

Webcam Subir

Preparación

Preparar modelo

Avanzado

Vista previa

Para obtener una vista previa de un modelo aquí, primero debes prepararlo en la parte de la izquierda.

Exportar modelo

Añadir una clase

español release-2-4-10 - 2.4.10#40c178

≡ Teachable Machine

Frente

65 muestras de imágenes

Webcam Subir

Perfil

54 muestras de imágenes

Webcam Subir

Preparación

Preparando...

Disponiendo datos de preparación...

Avanzado

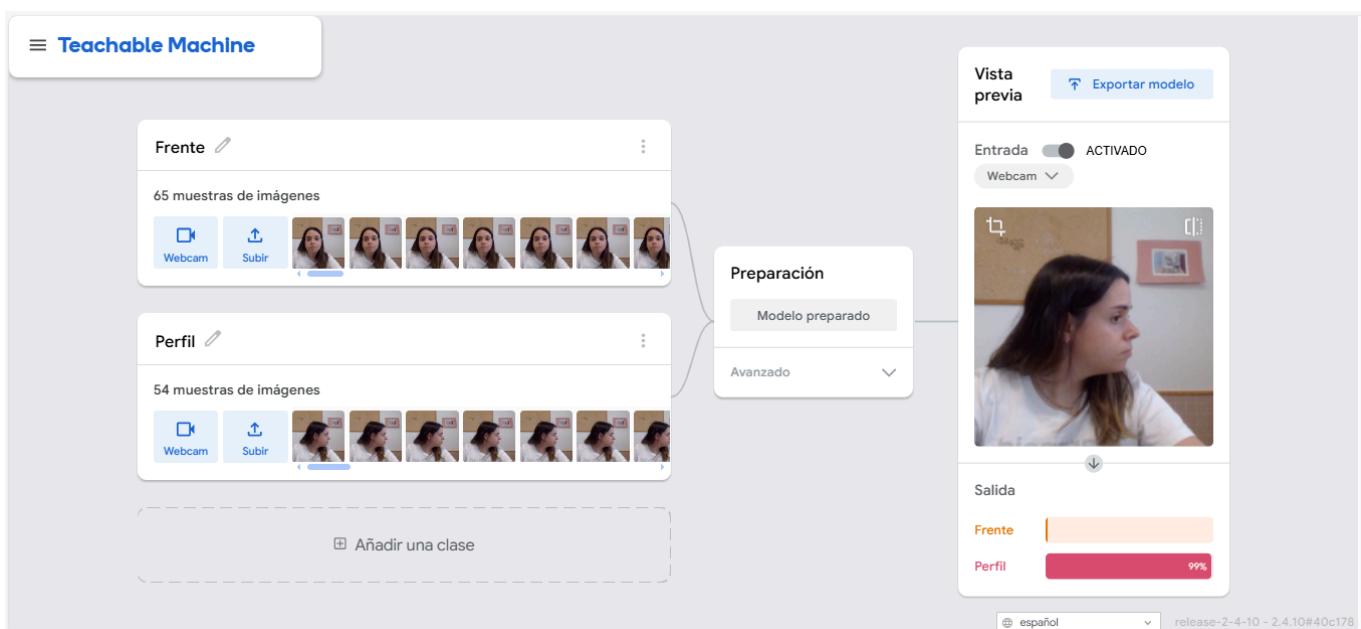
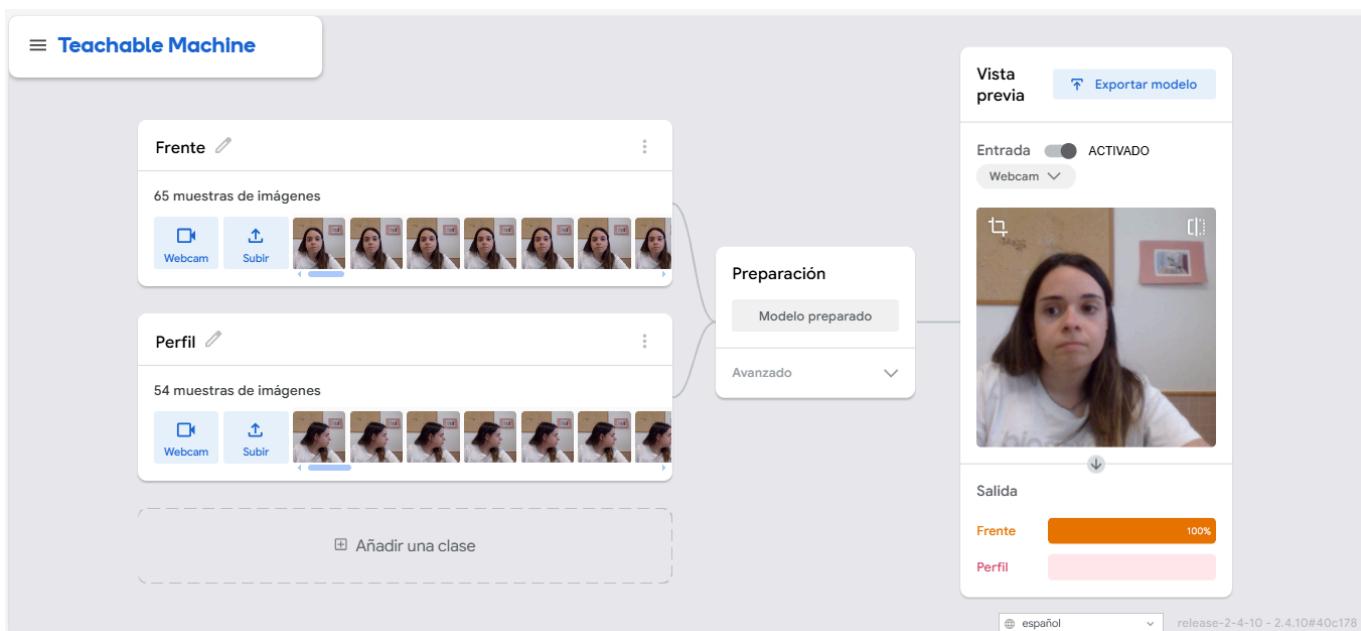
Vista previa

Para obtener una vista previa de un modelo aquí, primero debes prepararlo en la parte de la izquierda.

Exportar modelo

Añadir una clase

español release-2-4-10 - 2.4.10#40c178



Exportar modelo a TensorFlow (.JS)

Cuando terminamos el modelo en **Teachable Machine** y elegimos exportarlo para la web, nos da un modelo compatible con *TensorFlow.js*. Esto incluye:

- Dos archivos **model** y **metadata** (.json).
- Un archivo **weights** (.bin)
- Y un pequeño ejemplo en *JavaScript*:

Exportar el modelo para usarlo en proyectos.

Tensorflow.js ⓘ Tensorflow ⓘ Tensorflow Lite ⓘ

Exporta tu modelo:

Subir (enlace para compartir) Descargar [!\[\]\(2fddc4b42a48b1de52379751b2a31e0f_img.jpg\) Descargar modelo](#)

Fragmentos de código para usar el modelo:

[Javascript](#) [p5.js](#) [Contribuye en Github](#) 

Learn more about how to use the code snippet on [github](#).

```
<div>Teachable Machine Image Model</div>
<button type="button" onclick="init()">Start</button>
<div id="webcam-container"></div>
<div id="label-container"></div>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@latest/dist/teachablemachine-image.min.js"></script>
<script type="text/javascript">
    // More API functions here:
    // https://github.com/googlecreativelab/teachablemachine-community/tree/master/libraries/image

    // the link to your model provided by Teachable Machine export panel
    const URL = "./my_model/";

    let model, webcam, labelContainer, maxPredictions;

    // Load the image model and setup the webcam
    async function init() {
        const modelURL = URL + "model.json":
```

[Copiar](#) 

```
<div>Teachable Machine Image Model</div>
<button type="button" onclick="init()">Start</button>
<div id="webcam-container"></div>
<div id="label-container"></div>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@latest/dist/teachablemachine-image.min.js"></script>
<script type="text/javascript">
    // More API functions here:
```

```

// https://github.com/googlecreativelab/teachablemachine-community/

// the link to your model provided by Teachable Machine export pane
const URL = "./my_model/";

let model, webcam, labelContainer, maxPredictions;

// Load the image model and setup the webcam
async function init() {
    const modelURL = URL + "model.json";
    const metadataURL = URL + "metadata.json";

    // load the model and metadata
    // Refer to tmImage.loadFromFiles() in the API to support files
    // or files from your local hard drive
    // Note: the pose library adds "tmImage" object to your window
    model = await tmImage.load(modelURL, metadataURL);
    maxPredictions = model.getTotalClasses();

    // Convenience function to setup a webcam
    const flip = true; // whether to flip the webcam
    webcam = new tmImage.Webcam(200, 200, flip); // width, height,
    await webcam.setup(); // request access to the webcam
    await webcam.play();
    window.requestAnimationFrame(loop);

    // append elements to the DOM
    document.getElementById("webcam-container").appendChild(webcam);
    labelContainer = document.getElementById("label-container");
    for (let i = 0; i < maxPredictions; i++) { // and class labels
        labelContainer.appendChild(document.createElement("div"));
    }
}

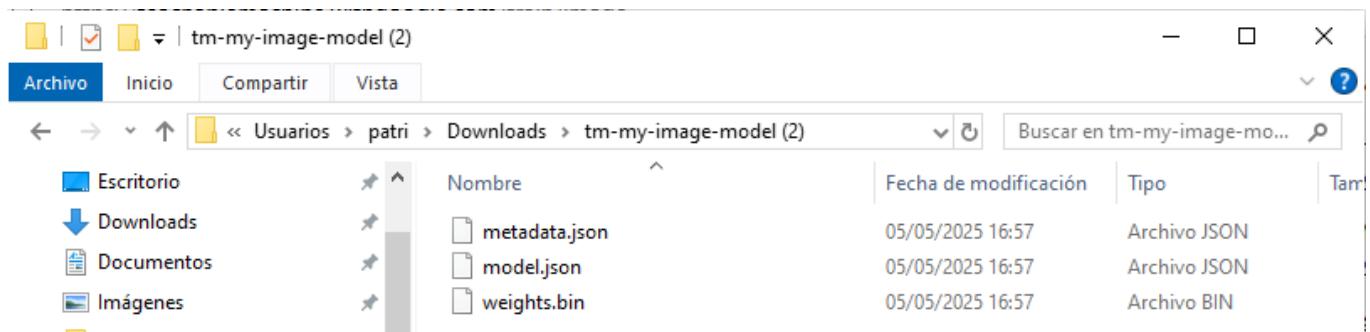
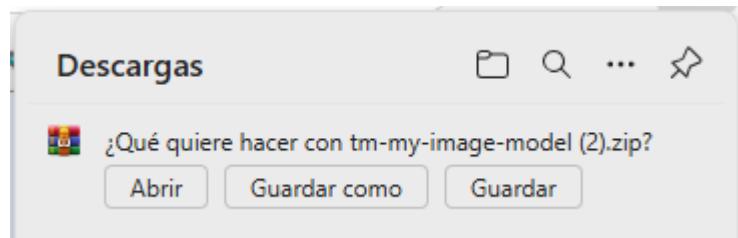
async function loop() {
    webcam.update(); // update the webcam frame
    await predict();
    window.requestAnimationFrame(loop);
}

// run the webcam image through the image model
async function predict() {
    // predict can take in an image, video or canvas html element
    const prediction = await model.predict(webcam.canvas);
    for (let i = 0; i < maxPredictions; i++) {
        const classPrediction =

```

```
56         prediction[i].className + ": " + prediction[i].probabil
57         labelContainer.childNodes[i].innerHTML = classPrediction;
58     }
59 }
```

```
</script>
```



Embeber modelo en web (HTML+JS)

⚠ Muy importante para tu app con *Teachable Machine*:

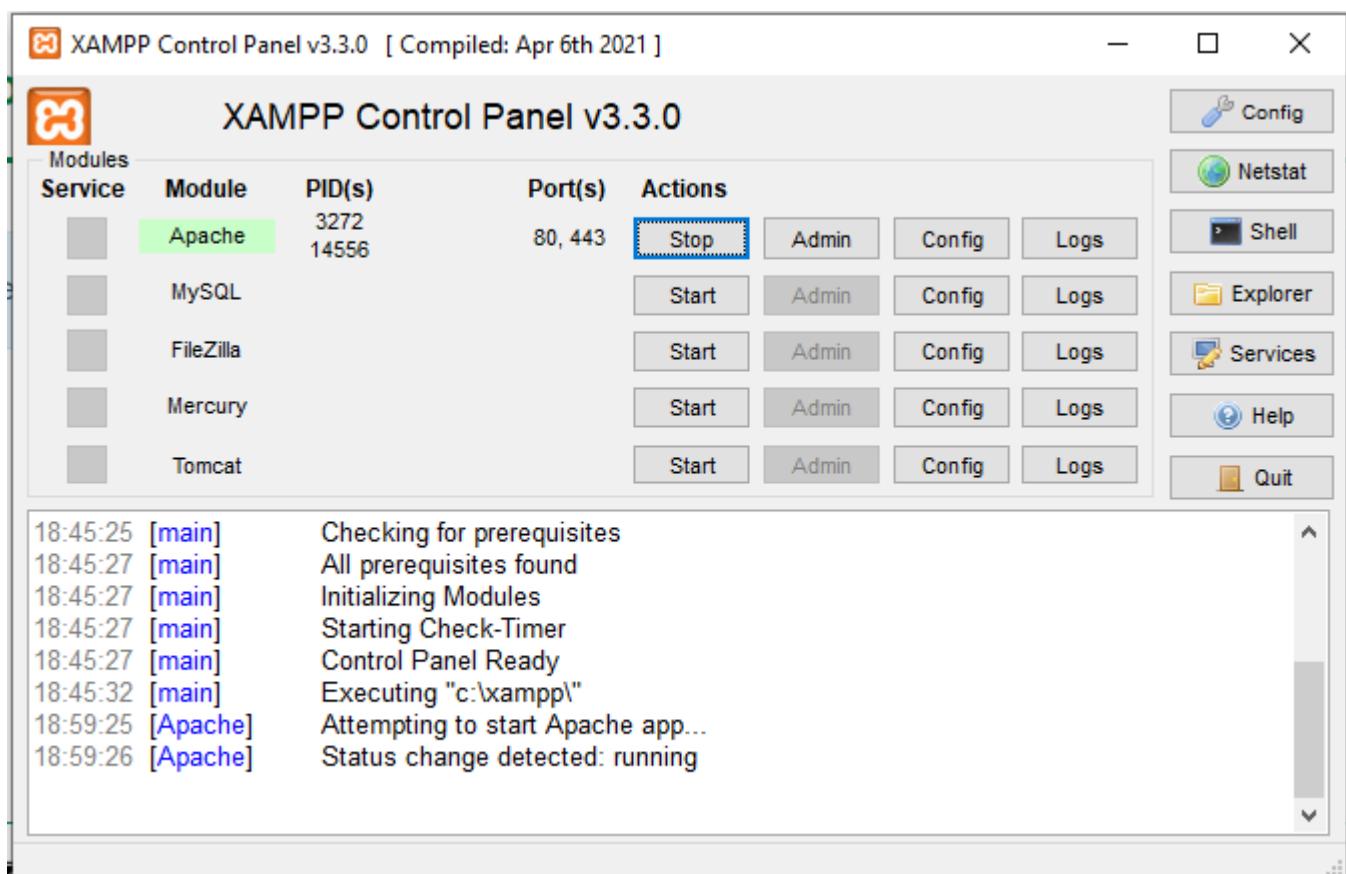
Cuando trabajemos con archivos locales de modelos (.json, .bin) necesitaremos un servidor (aunque sea *localhost*) porque los navegadores:

- No permiten cargar archivos file:// directos por seguridad.
- No permiten acceso a la cámara en páginas abiertas como file://.

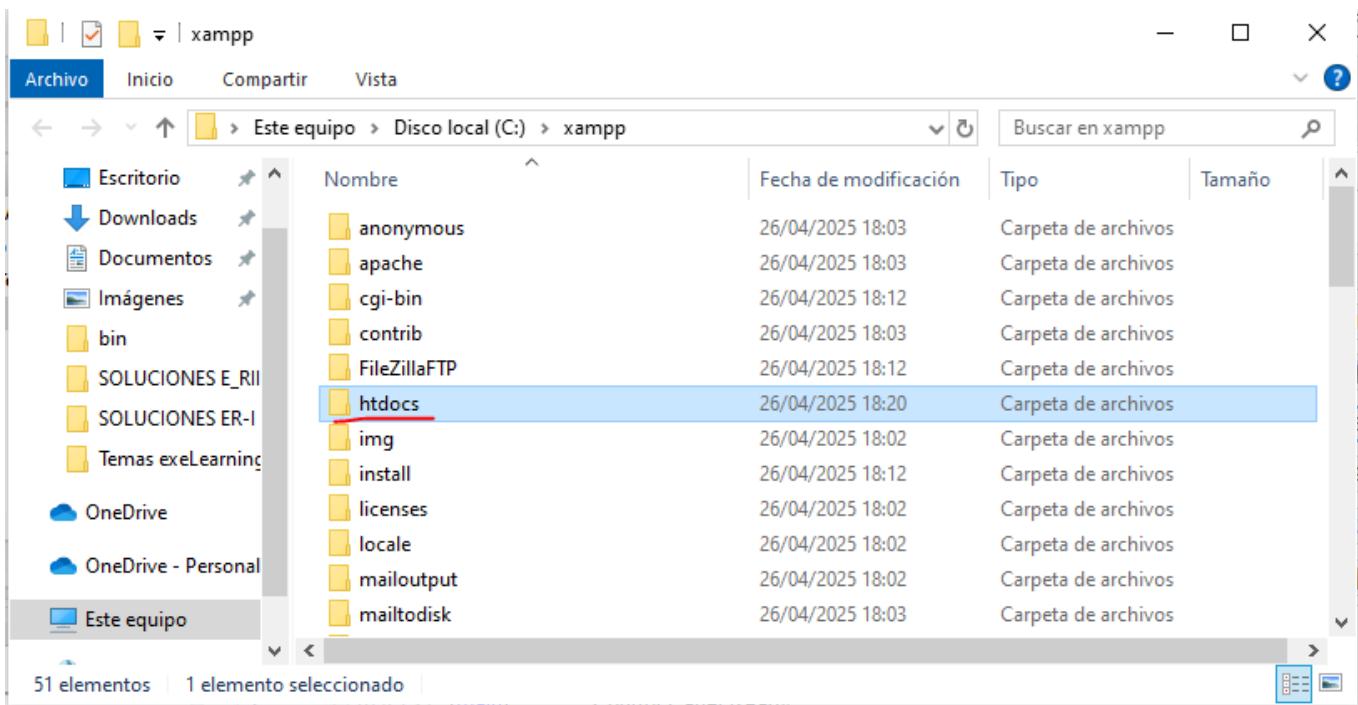
Con *XAMPP*, solucionamos todo eso.

Teniendo esto en cuenta, para integrarlo en una app real, el flujo básico sería:

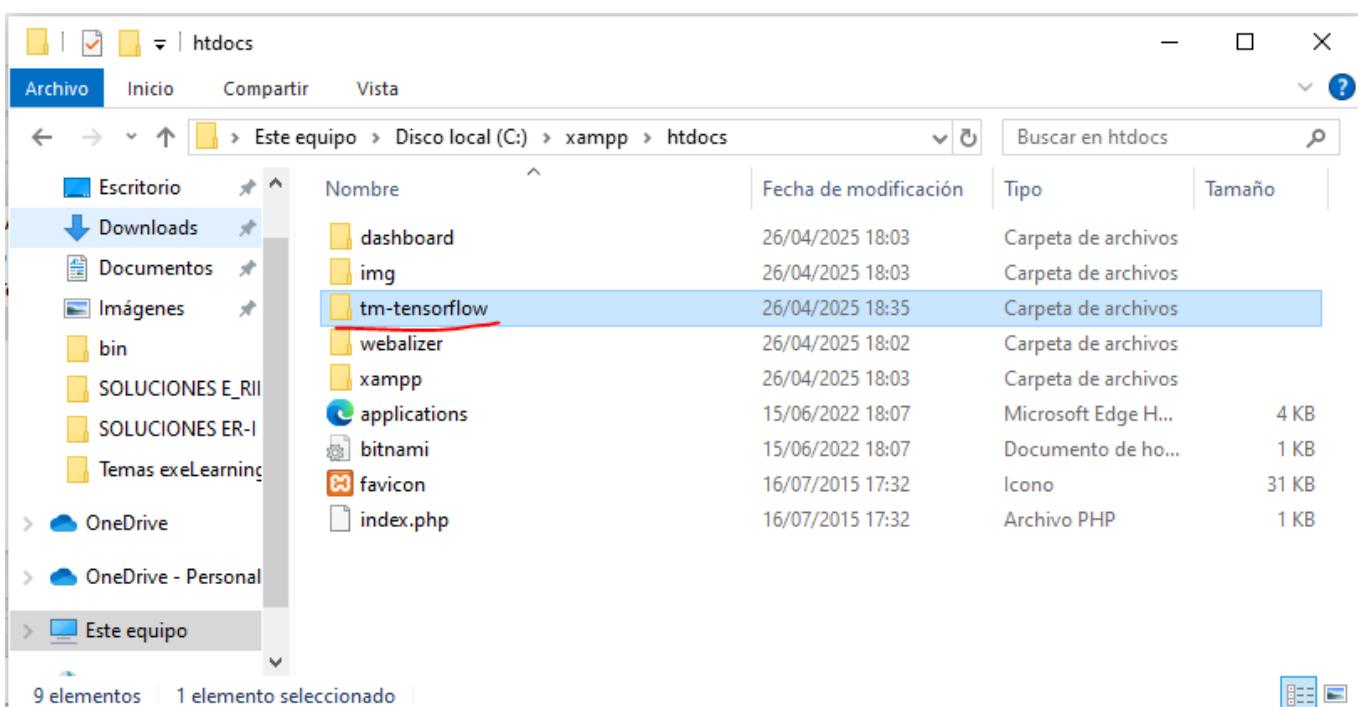
- Subir el modelo (los .json y los .bin) a un servidor o almacenamiento online (o puedes guardarlo localmente en tu proyecto). Como ya hemos dicho, nosotros usaremos el servidor de *Apache* de *XAMPP*.



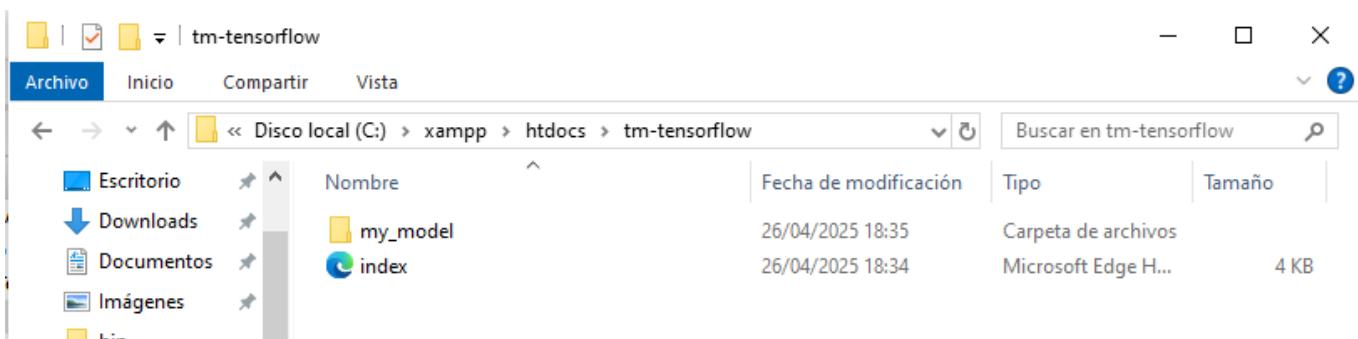
- Accede al explorador de archivos de *XAMPP* y busca la carpeta *htdocs*.



- Dentro de *htdocs*, crea una carpeta para tu proyecto. Para este ejemplo la llamaremos ***tm-tensorflow***.



- Dentro de ***tm-tensorflow*** deberemos pegar la carpeta descargada desde *Teachable Machine*, que contiene el .json y el .bin.



- Además, en esa misma carpeta, deberemos añadir un `index.HTML` que embeda el código *Javascript* proporcionado también por la herramienta. Puedes usar este de ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Teachable Machine con Webcam</title>

    <!-- Librerías necesarias -->
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js">
    <script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@1.0.1/dist/teachable-machine.js">

    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        video {
            margin-top: 20px;
            border: 2px solid #ccc;
            border-radius: 10px;
        }
        #start-button {
            padding: 10px 20px;
            font-size: 18px;
            margin-bottom: 20px;
            cursor: pointer;
        }
        #prediction {
            margin-top: 20px;
            font-size: 20px;
        }
        #webcam-container {
            margin-top: 20px;
        }
    </style>
</head>

<body>

    <h1>Teachable Machine Web App</h1>
```

```

<!-- Botón para iniciar -->
<button id="start-button" type="button" onclick="init()">Iniciar Predic

<div id="webcam-container"></div>
<div id="label-container"></div>

<script type="text/javascript">
  const URL = "./my_model/"; // Asegúrate que esta sea la ruta correcta

  let model, webcam, labelContainer, maxPredictions;

  // Función para cargar el modelo y configurar la webcam
  async function init() {
    const modelURL = URL + "model.json";
    const metadataURL = URL + "metadata.json";

    // Cargar el modelo y los metadatos
    model = await tmImage.load(modelURL, metadataURL);
    maxPredictions = model.getTotalClasses();

    // Configuración de la webcam
    const flip = true; // voltear la cámara
    webcam = new tmImage.Webcam(200, 200, flip); // Configurar webcam
    await webcam.setup(); // Solicitar acceso a la webcam
    await webcam.play();
    window.requestAnimationFrame(loop);

    // Insertar el video de la cámara en el DOM
    document.getElementById("webcam-container").appendChild(webcam.ca

    // Preparar el contenedor para las predicciones
    labelContainer = document.getElementById("label-container");
    for (let i = 0; i < maxPredictions; i++) {
      labelContainer.appendChild(document.createElement("div"));
    }
  }

  // Función para predecir continuamente
  async function loop() {
    webcam.update(); // Actualizar el cuadro de la cámara
    await predict();
    window.requestAnimationFrame(loop);
  }

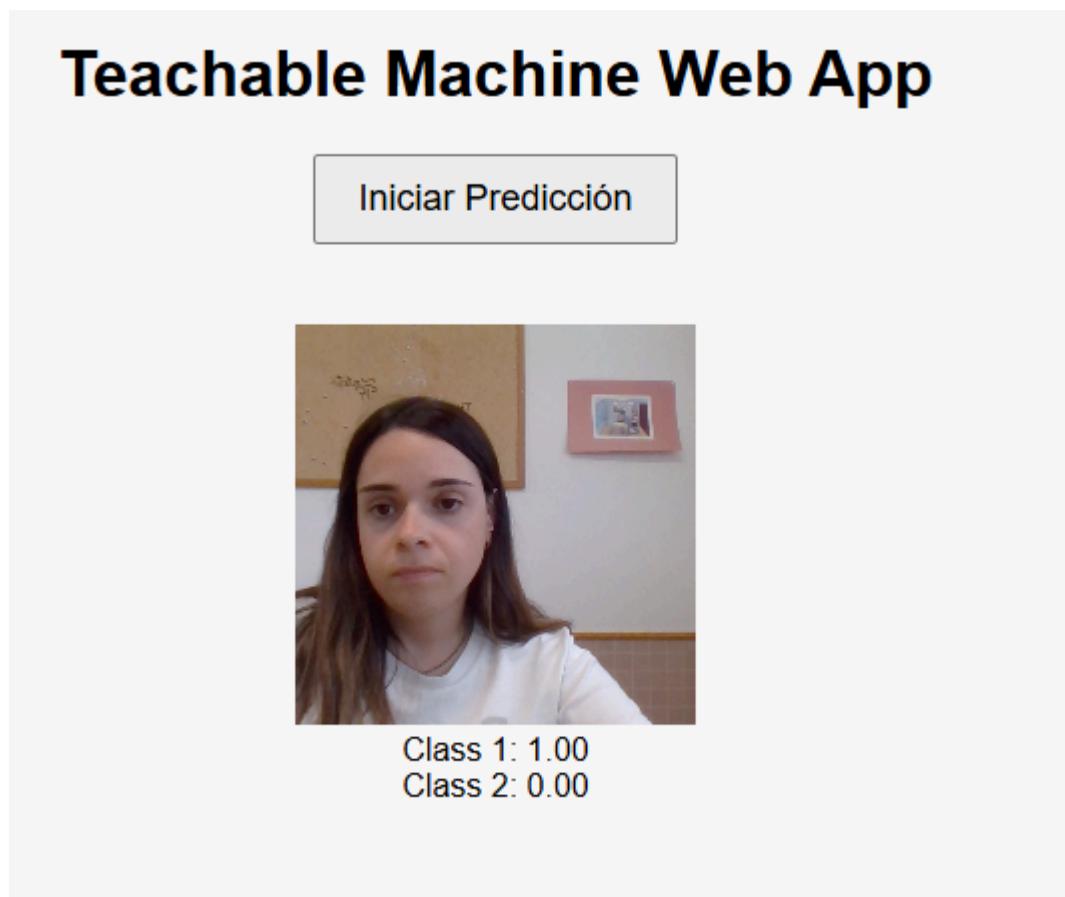
  // Función de predicción usando el modelo
  async function predict() {

```

```
88     const prediction = await model.predict(webcam.canvas); // Ejecuta
89     for (let i = 0; i < maxPredictions; i++) {
90         const classPrediction = prediction[i].className + ": " + predic
91         labelContainer.childNodes[i].innerHTML = classPrediction; // Mo
92     }
93 }
94 </script>
95
96 </body>
97 </html>
```

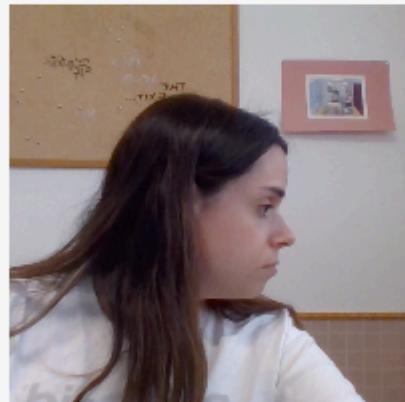
- Accede desde cualquier navegador a la url de tu servidor (*localhost*), dirígete a la ruta */tm-tensorflow* y comprueba que la app funciona como esperamos.

<http://localhost/tm-tensorflow/> <<http://localhost/mi-app-tm/>>



Teachable Machine Web App

Iniciar Predicción



Class 1: 0.27
Class 2: 0.73

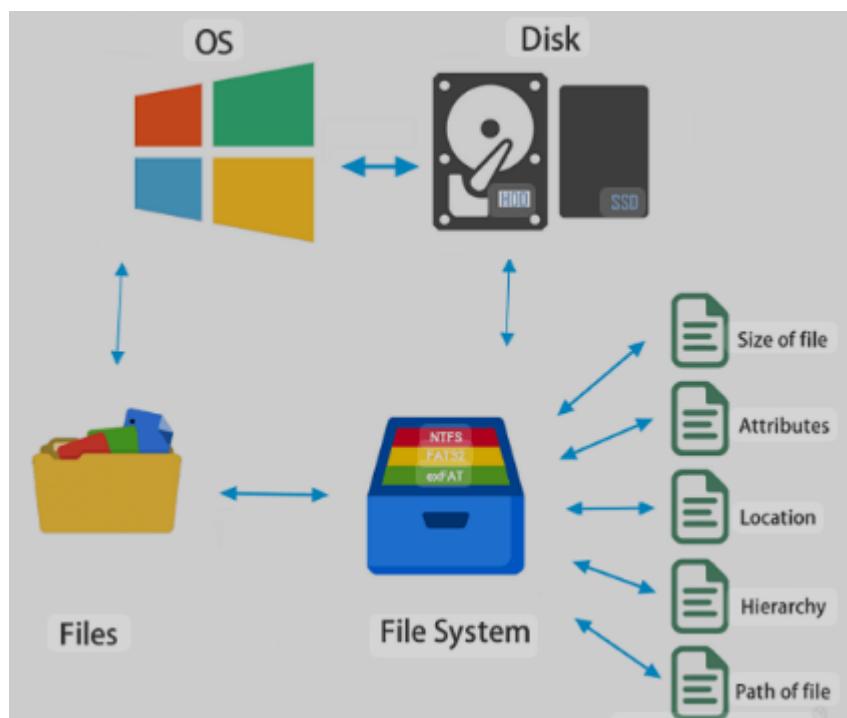


Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<http://creativecommons.org/licenses/by-sa/4.0/>

Ficheros

8.3. Ficheros y flujos en Java

Un fichero (o archivo) se refiere a una secuencia de datos almacenados en un sistema de archivos en un dispositivo de almacenamiento, como un disco duro o una unidad USB. Los ficheros pueden contener información en forma de texto, imágenes, audio, vídeo u otros formatos de datos.



En **Java**, los ficheros pueden clasificarse en dos categorías principales: ficheros de texto y ficheros binarios. La distinción entre estos dos tipos de ficheros radica en la forma en que se almacenan y se procesan los datos.

Ficheros de Texto

- Secuencia de caracteres
- Interpretable por un ser humano
- Generalmente portable
- Escritura/Lectura menos eficiente que los ficheros binarios
- Requiere más tamaño que un fichero binario para representar la misma información
- Ej: Un entero de 9 dígitos en un fichero de texto ocuparía 18 bytes (asumiendo codificación Unicode de 2 bytes/carácter)

Ficheros Binarios

- Secuencia de bytes (interpretados como tipos primitivos)
- No interpretable por un ser humano
- Generalmente no portable (debe ser leído en el mismo tipo de ordenador y con el mismo lenguaje de programación que fue escrito)
- Escritura/Lectura eficiente
- Almacenamiento eficiente de la información
- Ej: Un entero de 9 dígitos en un fichero binario ocupa 4 bytes

Podemos usar Java para:

- Leer datos de un archivo.
- Escribir datos en un archivo.
- Crear o eliminar archivos.
- Verificar si existe un archivo o directorio.

Existen varias clases para trabajar con archivos, principalmente en el paquete **java.io** y **java.nio.file**.

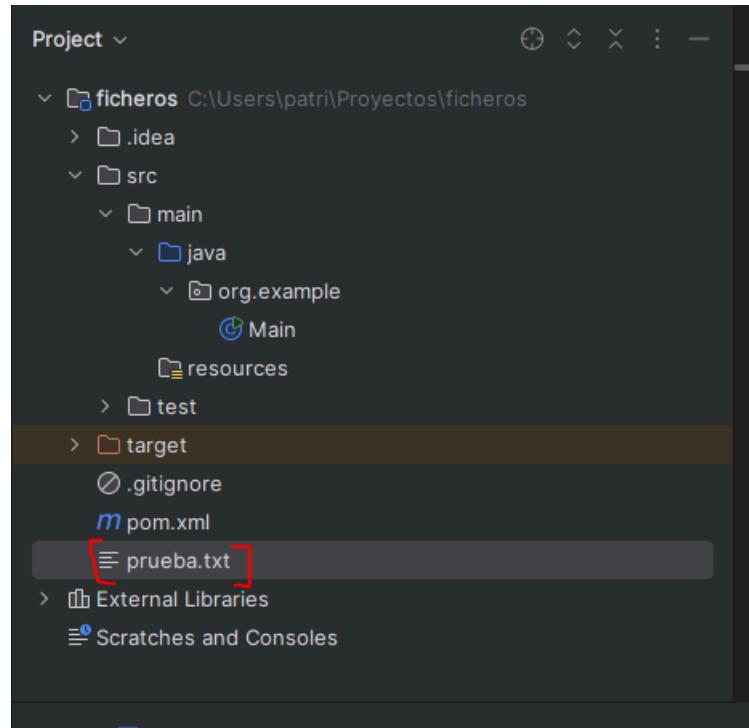
8.3.1. Clase File

La clase **File** no se utiliza para leer o escribir datos en sí, sino para acceder a la información del archivo o directorio, como la ruta, tamaño, atributos, permisos, etc.

```
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         File archivo = new File("prueba.txt"); // archivo es una refere  
5 //No crea físicamente el archivo prueba.txt aún, sólo representa una ru  
6  
7         try {  
8             if (archivo.createNewFile()) {  
9                 System.out.println("Archivo creado: " + archivo.getName());  
10            } else {  
11                System.out.println("El archivo " + archivo.getName() +  
12                );  
13            } catch (IOException e) {  
14                System.out.println("Ha habido algún problema.");  
15                e.printStackTrace();  
16            }  
17        }  
18    }  
19}
```

Como nos interesa saber qué ha ocurrido si se ha producido un error, en lugar de `e.getMessage()` usaremos `e.printStackTrace()` que nos proporciona información sobre el tipo de excepción y la pila de llamadas (*stack trace*), es decir, el error que ha ocurrido en el código, línea por línea.

Por defecto, si no indicamos ruta, nos creará el archivo colgando de la carpeta principal del proyecto:



ACTIVIDAD 1. Modifica el programa para que los archivos se creen en la carpeta `src/main/resources/` a partir de ahora.

Método/Constructor	Descripción
<code>File(String pathname)</code>	Crea un nuevo objeto de tipo File a partir de su ruta.
<code>boolean createNewFile()</code>	Crea un nuevo archivo vacío con la ruta definida por el File
<code>boolean delete()</code>	Borra el fichero/directorio
<code>boolean exists()</code>	Indica si el fichero/directorio existe
<code>String getName()</code>	Devuelve el nombre del fichero/directorio (sin la ruta)
<code>String getParent()</code>	Devuelve la ruta al fichero/directorio padre
<code>File[] listFiles()</code>	Obtiene un listado de ficheros en el directorio
<code>boolean isDirectory()</code>	Indica si se trata de un directorio
<code>boolean isFile()</code>	Indica si se trata de un fichero
<code>getAbsolutePath()</code>	Retorna la ruta absoluta del archivo o directorio
<code>mkdir() y mkdirs()</code>	Crea un directorio/directorios representado por el File

```

File fichero = new File("src/main/resources/ejemplo1.txt");
if(fichero.exists()) System.out.println("El fichero " + fichero
else System.out.println("El fichero " + fichero.getName() + " n
System.out.println("Nombre: " + fichero.getName());
System.out.println("Longitud: " + fichero.length());
System.out.println("Ruta absoluta: " + fichero.getAbsolutePath()

```

```

8 // ejemplo carpeta
9     File carpeta = new File("src/main/resources");
10    if(carpeta.exists()) System.out.println("La carpeta " + carpeta
11    else System.out.println("La carpeta " + carpeta.getName() + " n
12    System.out.println("Nombre: " + carpeta.getName());
13    System.out.println("Longitud: " + carpeta.length());
14    System.out.println("Ruta absoluta: " + carpeta.getAbsolutePath())

```

```

El fichero no existe
Nombre: ejemplo1.txt
Longitud: 0
Ruta absoluta: C:\Users\patri\Proyectos\ficheros\src\main\resources\ejemplo1.txt
La carpeta existe
Nombre: resources
Longitud: 0
Ruta absoluta: C:\Users\patri\Proyectos\ficheros\src\main\resources
Process finished with exit code 0

```

ACTIVIDAD 2. Añade lógica al ejemplo anterior para que se cree el fichero "ejemplo1.txt" y observa que cambia la salida porque ahora lo encuentra.

ACTIVIDAD 3. Abre el fichero manualmente, escribe algo y vuelve a ejecutar tu programa. Observa que haya cambiado la longitud según el número de caracteres que hayas escrito.

Crear y borrar ficheros y carpetas

```

1 File archivo = new File("nuevoArchivo.txt");
2 try {
3     if (archivo.createNewFile()) System.out.println("Archivo creado");
4     else System.out.println("El archivo ya existe");
5 } catch (IOException e) { e.printStackTrace(); }
6
7 File directorio = new File("nuevoDirectorio");
8 if (directorio.mkdir()) System.out.println("Directorio creado");
9 else System.out.println("No se pudo crear el directorio");
10
11 File archivoBorrar = new File("archivoParaEliminar.txt");
12 if (archivoBorrar.delete()) System.out.println("Archivo eliminado");
13 else System.out.println("No se pudo eliminar el archivo");

```

ACTIVIDAD 4. Crea un método que debe generar ‘n’ archivos: *nombre(1).txt*, *nombre(2).txt*,... *nombre(n).txt* en la carpeta que se solicite al usuario por pantalla.

Listar archivos de un directorio

- Sólo nombre (*String*):

```
1  File directorio2 = new File(".");
2
3  String[] archivos = directorio2.list();
4
5  if (archivos != null && archivos.length > 0){
6      for (String a : archivos){
7          System.out.println(a);
8      }
9  }else{
10     System.out.println("No hay archivos en la carpeta");
11 }
```

- Nombre y + info (*File*):

```
public static void main(String[] args) {
    String rutaCarpeta = ".";
    listarArchivos(rutaCarpeta);
}

// método que lista todos los archivos de un carpeta, indicando el tam
public static void listarArchivos(String rutaCarpeta) {
    File carpeta = new File(rutaCarpeta);
    if (carpeta.isDirectory()) {
        File[] archivos = carpeta.listFiles();
        if (archivos != null && archivos.length > 0) {
            for (File f : archivos) {
                if (f.isFile()) {
                    System.out.println(f.getName() + " - " + f.length())
                }
            }
        }else{
            System.out.println("No se ha encontrado ningún archivo.");
        }
    }
}
```

```
        } else System.out.println("La ruta proporcionada no es una carpeta"
    }
```

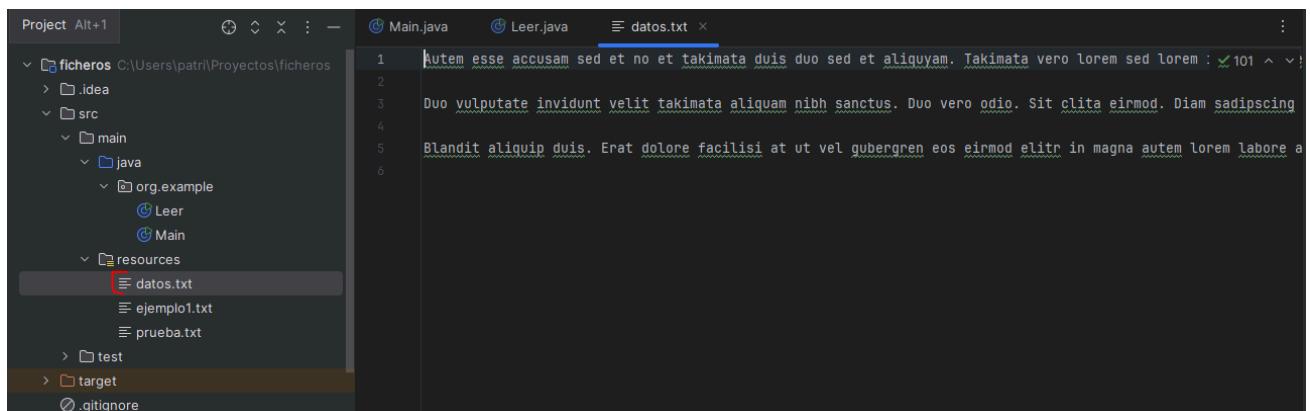
ACTIVIDAD 5. Crea un método que reciba una carpeta y liste el contenido de dicha carpeta de aquellos archivos cuya extensión sea *.txt*. Crea una sobrecarga para que el método pueda recibir también el tipo de archivo a listar (*.pdf*, *.jpg*, etc,...).

8.3.2. FileReader y BufferedReader (para leer archivos de texto)

FileReader se utiliza para leer caracteres en un archivo de texto. Convierte los caracteres a bytes utilizando la codificación de caracteres predeterminada del sistema y escribe esos bytes en el archivo.

BufferedReader se utiliza para leer el texto de manera eficiente. Internamente, utiliza un búfer (una región de memoria temporal) para almacenar datos de entrada. Al leer, **BufferedReader** intentará leer tantos caracteres como sea posible a la vez en el búfer, lo que puede ser mucho más eficiente que leer un carácter a la vez.

(!) Para llenar archivos de texto sin pensar mucho, utiliza esta herramienta [Lorem Ipsum TXT - Generate ipsum TXT file online <https://products.groupdocs.app/es/lorem-ipsum/txt>](https://products.groupdocs.app/es/lorem-ipsum/txt) para generar *Lorem Ipsum* de prueba. Genera un "*datos.txt*" y guárdalo en la carpeta */resources*.



Vamos a leerlo:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class EjemploLeer {

    public static void main(String[] args) {

        try {
            BufferedReader lector = new BufferedReader(new FileReader("resources/datos.txt"));
            String linea;
            while ((linea = lector.readLine()) != null) {
                System.out.println(linea);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

13     String linea;
14
15     while ((linea = lector.readLine()) != null) {
16         System.out.println(linea);
17     }
18
19     lector.close();
20
21 } catch (IOException e) {
22     System.out.println(e.getStackTrace());
23     throw new RuntimeException(e);
24 }
25
26 }
27
}

```

```

"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.2\lib\idea_rt.jar=55680:C:\Progr
Autem esse accusam sed et no et takimata duis duo sed et aliquyam. Takimata vero lorem sed lorem in lorem. Hendrerit in accusam sadipscing ea dolore magna
Duo vulputate invidunt velit takimata aliquam nibh sanctus. Duo vero odio. Sit clita eirmod. Diam sadipscing lorem magna et rebum lorem erat lorem lorem n
Blandit aliquip duis. Erat dolore facilisi at ut vel gubergren eos eirmod elitr in magna autem lorem labore amet exerci lorem erat. Accusam accumsan moles
Process finished with exit code 0

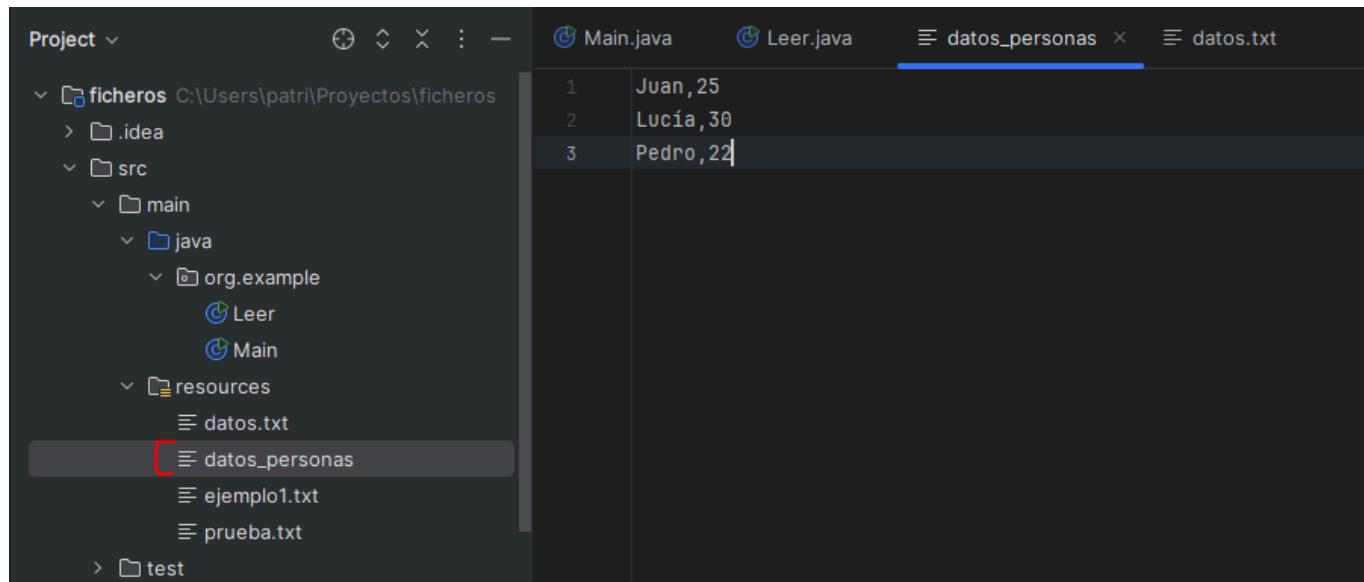
```

ACTIVIDAD 6. Crea un método que reciba una palabra y permita buscarla en un fichero de texto. Se debe mostrar el número de veces que aparece en el fichero dicha palabra.

Lección 10 - Entrada/Salida de datos

Nuestra conocida **Scanner** permite leer datos línea por línea, palabra por palabra o incluso por tipo (*int*, *double*, etc.).

- Supón que tenemos otro archivo llamado **datos_personas.txt** con este contenido, y queremos **leer línea por línea**:



```
1 | Juan,25
2 | Lucía,30
3 | Pedro,22
```

```
1 import java.io.File;
2 import java.io.FileNotFoundException;
3 import java.util.Scanner;
4
5 public class LeerArchivoConScanner {
6     public static void main(String[] args) {<br>
7         try {<br>
8             File archivo = new File("src/main/resources/datos_personas");
9             Scanner lector = new Scanner(archivo);
10
11             while (lector.hasNextLine()) {
12                 String linea = lector.nextLine();
13                 System.out.println("Línea: " + linea);
14             }
15
16             lector.close();<br>
17         } catch (FileNotFoundException e) {
18             System.out.println("Archivo no encontrado.");
19             e.printStackTrace();
20         }
21     }
22 }
```

```
Línea: Juan,25  
Línea: Lucía,30  
Línea: Pedro,22  
  
Process finished with exit code 0
```

- **Lectura por separador (,) (dato a dato):**

```
1 | while (lector.hasNextLine()) {  
2 |     String linea = lector.nextLine();  
3 |     String[] partes = linea.split(",");  
4 |     String nombre = partes[0];  
5 |     int edad = Integer.parseInt(partes[1]);  
6 |  
7 |     System.out.println(nombre + " tiene " + edad + " años.");  
8 | }
```

```
Juan tiene 25 años.  
Lucía tiene 30 años.  
Pedro tiene 22 años.  
  
Process finished with exit code 0
```

Si no existe separador explícito (espacios), le indicamos " " (nada) al `.split()`.

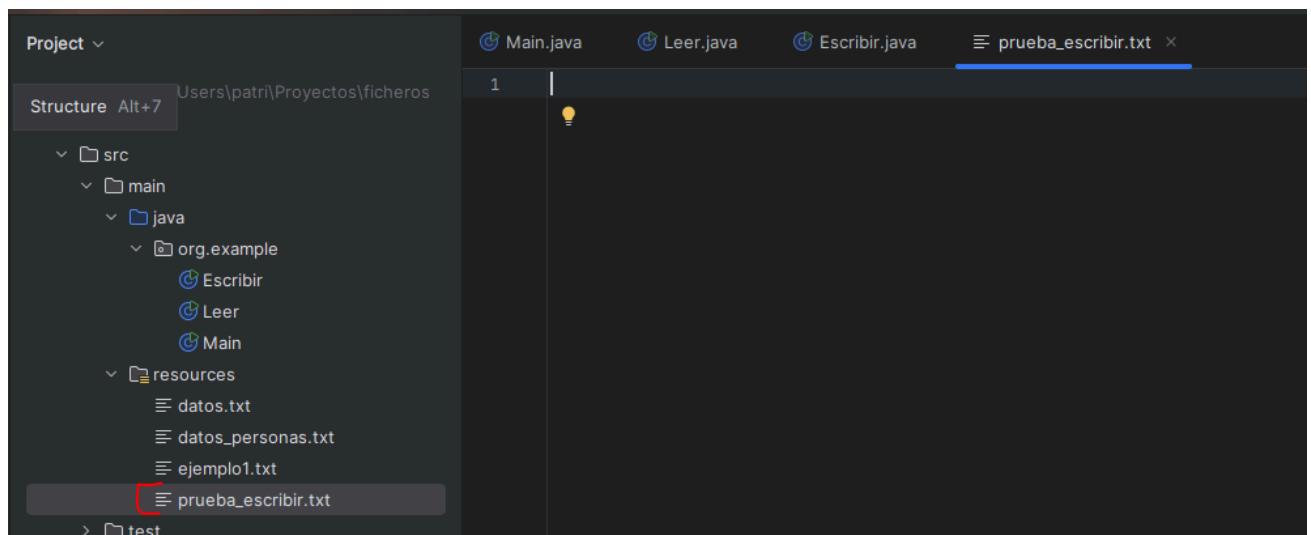
ACTIVIDAD 7. Crea un método que haga la misma función que el realizado en la **ACTIVIDAD 6 (contar palabras)**, pero que vaya leyendo el fichero palabra por palabra con la clase `Scanner`.

8.3.3. **FileWriter** y **BufferedWriter** (para escribir archivos de texto)

FileWriter se utiliza para escribir caracteres desde un archivo de texto. Se encarga de convertir los bytes del archivo a caracteres utilizando la codificación de caracteres predeterminada del sistema.

BufferedWriter se utiliza para escribir texto de manera eficiente. Al igual que **BufferedReader**, **BufferedWriter** utiliza un búfer para almacenar datos de salida.

Para probar, crearemos un fichero nuevo llamado "**prueba_escribir.txt**":



```
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.IOException;

public class Escribir {

    public static void main(String[] args) {

        try {

            BufferedWriter escritor = new BufferedWriter(new FileWriter("Hola, mundo!"));
            escritor.newLine();
            escritor.write("Esto se guarda en el archivo.");

            escritor.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

19     } catch (IOException e) {
20         System.out.println("Ha habido algún problema.");
21         e.printStackTrace();
22     }
23 }
24 }
25 }
```

The screenshot shows a dark-themed code editor with several tabs at the top: Main.java, Leer.java, Escribir.java, and prueba_escribir.txt. The prueba_escribir.txt tab is active, showing its content in a monospaced font. The content consists of two lines: 'Hola, mundo!' and 'Esto se guarda en el archivo.'. The second line ends with a cursor.

Cuando escribimos en *BufferedWriter* los datos se escriben primero en el búfer y se vacían (escriben) cuando el búfer está lleno, o cuando se usa el método *flush()*, que fuerza la escritura inmediata del contenido del buffer al archivo, sin necesidad de cerrar el flujo:

```

1 try {
2     BufferedWriter writer = new BufferedWriter(new FileWriter(
3
4         writer.write("Primera línea escrita.");
5         writer.newLine();
6
7         // fuerza la escritura inmediata del contenido al archivo
8         writer.flush();
9         System.out.println("Primera línea escrita y guardada (flush");
10
11        // seguimos escribiendo
12        writer.write("Segunda línea escrita.");
13        writer.newLine();
14
15        // finalmente cerramos (esto también hace flush implícitamente)
16        writer.close();
17        System.out.println("Segunda línea escrita y archivo cerrado");
18
19    } catch (IOException e) {
20        System.out.println("Ha habido algún problema.");
21        e.printStackTrace();
22    }
23 }
```

ACTIVIDAD 8. Modifica el método creado en la ACTIVIDAD 2 para que dentro de cada archivo generado se escriba la frase “*Este es el fichero nombre(n).txt*”.

ACTIVIDAD 9. Crea un método que reciba un archivo de texto y modifique su contenido, de modo que cada palabra del archivo deberá empezar en mayúscula.

(!) **PISTA** Ve leyendo el archivo original y escribiendo las modificaciones sobre un archivo temporal:

```
1 | BufferedReader reader = new BufferedReader(new FileReader(archivo));
2 | BufferedWriter writer = new BufferedWriter(new FileWriter(archivoTemp))
```

Cuando termines, reemplaza el archivo original por el modificado:

```
1 |         // reemplazar el archivo original por el modificado
2 |         if (archivo.delete()) {
3 |             archivoTemp.renameTo(archivo);
4 |             System.out.println("Archivo modificado correctamente.");
5 |         } else {
6 |             System.out.println("No se ha podido reemplazar el archivo o")
7 |         }
```

La **clase PrintWriter** es una subclase de la clase abstracta *Writer* y se utiliza para escribir datos formateados en un flujo de salida. A diferencia de *FileWriter*, tiene la capacidad de imprimir representaciones de varios tipos de datos, incluyendo texto y primitivas como *int*, *long*, *float*, etc. Además, proporciona métodos para imprimir líneas completas (*println()*), que pueden ser muy útiles. Por ello, *PrintWriter* es mucho más versátil que *FileWriter*, y más conveniente cuando quieras imprimir cosas como si usaras *System.out*.

```
import java.io.*;

public class EjemploPrintWriter {
    public static void main(String[] args) {
        try {
            PrintWriter writer = new PrintWriter(new FileWriter("salida
writer.println("Primera línea con println()");
writer.flush(); // fuerza la escritura inmediata
```

```
10     System.out.println("Primera línea escrita y guardada (flush  
11  
12     writer.printf("Número: %.2f\n", 3.1416);  
13     writer.println("Otra línea más.");  
14  
15     writer.close(); // flush() también ocurre automáticamente  
16     System.out.println("Archivo cerrado.");  
17  
18 } catch (IOException e) {  
19     System.out.println("Ha habido algún problema.");  
20     e.printStackTrace();  
21 }  
22 }  
23 }  
24 }
```

ACTIVIDAD 10. Crea un método que reciba 2 archivos de texto y combine el contenido de los 2 archivos. Para ello, se creará un nuevo archivo donde se debe añadir una palabra de cada archivo de forma consecutiva mientras queden palabras en cada uno de los archivos. Si algún archivo se queda sin palabras, se deben seguir añadiendo todas las palabras que quedan en el otro archivo.

8.3.4. Ficheros binarios

Un **fichero binario** almacena datos en formato no legible para humanos. A diferencia de los ficheros de texto, donde los datos se guardan como caracteres, en los binarios se almacenan en forma de bytes. Este formato es más compacto y eficiente en términos de espacio y tiempo de procesamiento, ya que no requiere conversiones adicionales.

Para trabajar en Java con ficheros binarios, se usan estas clases:

- **DataOutputStream**: para escribir datos binarios.
- **DataInputStream**: para leer datos binarios.

Estas clases se usan sobre un *FileOutputStream / FileInputStream*.

Para lectura o escritura de fichero binario:

1. Crear un **File** con el origen/destino de datos.
2. Envolverlo en un *FileInputStream/FileOutputStream* para crear un flujo de datos *desde/hacia* el fichero.
3. Envolver el objeto anterior en un *DataInputStream/DataOutputStream* para poder *leer/escribir* tipos de datos primitivos del flujo de datos.
4. Usar métodos del estilo *writeInt()*, *writeDouble()*, *readInt()*, *readDouble()*, etc.).

Ejemplo: escribir datos binarios

```
1 import java.io.*;  
2  
3 public class EscribirBinario {  
4     public static void main(String[] args) throws IOException {  
5         DataOutputStream out = new DataOutputStream(new FileOutputStream("binario.dat"));  
6  
7         out.writeInt(42);           // escribe un int  
8         out.writeDouble(3.14);      // escribe un double  
9         out.writeUTF("Hola");      // escribe una cadena  
10        out.writeBoolean(true);   // escribe un booleano  
11  
12        out.close();  
13        System.out.println("Datos binarios guardados.");  
14    }  
15 }
```

Ejemplo: leer datos binarios

Para poder leer el contenido de un fichero binario, debemos conocer la estructura interna del fichero. Es decir, debemos saber cómo se han escrito: si hay *enteros*, *long*, etc. y en qué orden están escritos en el fichero.

```
1 import java.io.*;  
2  
3 public class LeerBinario {  
4     public static void main(String[] args) throws IOException {  
5         DataInputStream in = new DataInputStream(new FileInputStream("d  
6  
7         int numero = in.readInt();  
8         double decimal = in.readDouble();  
9         String texto = in.readUTF();  
10        boolean bandera = in.readBoolean();  
11  
12        in.close();  
13  
14        System.out.println("Número: " + numero);  
15        System.out.println("Decimal: " + decimal);  
16        System.out.println("Texto: " + texto);  
17        System.out.println("Booleano: " + bandera);  
18    }  
19 }
```

Si no se conoce su estructura, podemos leerlo *byte* a *byte* con el método *.read()*. Lee un *byte* y lo devuelve como *int* (entre 0 y 255 porque un *byte* es de 8 bits). Si no puede leer, devuelve un -1.

```
1 String fileName = "datos.bin";  
2 // leer datos desde el archivo binario  
3 try {  
4     FileInputStream fis = new FileInputStream(fileName);  
5     int byteLeido;  
6     System.out.println("Datos leídos desde el archivo binario:");  
7     while ((byteLeido= fis.read()) != -1){  
8         System.out.print((char) byteLeido); // convertimos byte a car  
9     }  
10    fis.close();  
11 } catch (IOException e) { System.err.println("Error al leer"); }
```

Para guardar información de objetos completos, lo mejor es usar **serialización**. La vemos en el siguiente apartado del tema.

Práctica 3. Manejo de ficheros en Java

Recopila en un documento de texto las evidencias de funcionamiento de las actividades planteadas durante la parte teórica sobre ficheros. Súbelo a la entrega disponible en AULES.



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<<http://creativecommons.org/licenses/by-sa/4.0/>>

8.4. Serialización de objetos.

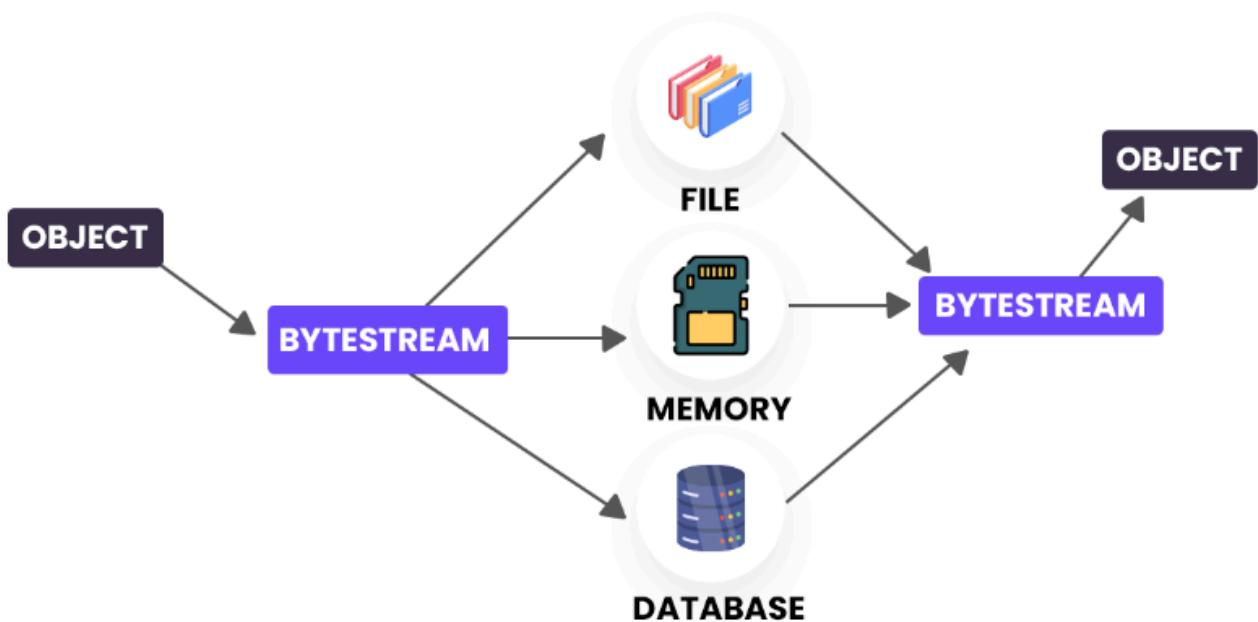
8.4. Serialización de objetos.

La **serialización** en Java es una técnica para guardar (y luego recuperar) el estado de un objeto en un fichero o en otro medio de almacenamiento, como si lo “congeláramos” y después lo “reviviéramos”.

La **deserialización** hace lo contrario: reconstruye el objeto original a partir de los *bytes* guardados.

Serialization

De-Serialization



Para poder usarlas, nuestras clases deben implementar la interfaz **Serializable**, aunque esta solamente sirve para marcar y no obliga a implementar ningún método (porque no tiene ninguno).

If a serializable class does not explicitly declare a serialVersionUID, then the serialization runtime will calculate a default serialVersionUID value for that class based on various aspects of the class, as described in the [Java Object Serialization Specification](#). This specification defines the serialVersionUID of an enum type to be 0L. However, it is *strongly recommended* that all serializable classes other than enum types explicitly declare serialVersionUID values, since the default serialVersionUID computation is highly sensitive to class details that may vary depending on compiler implementations, and can thus result in unexpected `InvalidClassException`s during deserialization. Therefore, to guarantee a consistent serialVersionUID value across different java compiler implementations, a serializable class must declare an explicit serialVersionUID value. It is also strongly advised that explicit serialVersionUID declarations use the `private` modifier where possible, since such declarations apply only to the immediately declaring class--serialVersionUID fields are not useful as inherited members. Array classes cannot declare an explicit serialVersionUID, so they always have the default computed value, but the requirement for matching serialVersionUID values is waived for array classes.

Since: 1.1

See Also: `ObjectOutputStream`,
`ObjectInputStream`,
`ObjectOutput`,
`ObjectInput`,
`Externalizable`

spec [serialization/index.html](#) Java Object Serialization Specification

```
192 public interface Serializable {  
193 }  
194
```

Clases `ObjectOutputStream` (serializar) y `ObjectInputStream` (deserializar)

Estas clases, igual que pasaba sobre la lectura y escritura de ficheros binarios, se usan a su vez sobre clases `FileOutputStream` ó `FileInputStream`.

Vamos a usar la siguiente clase `Persona` de referencia para instanciar objetos y realizar los ejemplos:

```
1 public class Persona implements Serializable {  
2     private String nombre;  
3     private int edad;  
4  
5     public Persona(String nombre, int edad) {  
6         this.nombre = nombre;  
7         this.edad = edad;  
8     }  
9  
10    public String toString() {  
11        return nombre + " tiene " + edad + " años.";  
12    }  
13}
```

- Serializamos (guardando el objeto en un archivo):

```

1 public class SerializarPersona {
2     public static void main(String[] args) {
3
4         Persona p = new Persona("Carlos", 30);
5
6         try {
7
8             ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("C:/Users/LENOVO/Desktop/persona.ser"));
9
10            out.writeObject(p);
11
12            out.close();
13
14        } catch (IOException e) {
15            System.out.println("Algo ha ido mal.");
16            e.printStackTrace();
17        }
18
19        System.out.println("Objeto serializado en persona.ser");
20
21    }
22}

```

The screenshot shows a Java development environment with the following details:

- Project View:** Shows the project structure with files: Leer, Main, Persona, and SerializarPersona. Under resources, there are files: datos.bin, datos.txt, datos_personas.txt, ejemplo1.txt, persona.ser (which is selected), and prueba_escribir.txt.
- Code Editor:** Displays the code for the `SerializarPersona` class.
- Run Tab:** Shows the command being run: `"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:/Users/LENOVO/Desktop/Leer.jar" -jar C:/Users/LENOVO/Desktop/SerializarPersona.jar`.
- Output Tab:** Displays the output of the program, showing the message: `Objeto serializado en persona.ser` and `Process finished with exit code 0`.

El archivo resultante (*persona.ser*) no es legible a simple vista: está en formato binario.

- Deserializamos (leemos el objeto desde el archivo creado anteriormente):

```
1 public class DeserializarPersona {  
2     public static void main(String[] args){  
3  
4         try {  
5             ObjectInputStream in = new ObjectInputStream(new FileInputStream("C:\\Users\\User\\Desktop\\persona.ser"));  
6             Persona p = (Persona) in.readObject();  
7             in.close();  
8             System.out.println("Objeto persona leído: " + p);  
9         } catch (IOException | ClassNotFoundException e) {  
10             System.out.println("Algo ha ido mal.");  
11             e.printStackTrace();  
12         }  
13     }  
14 }  
15 }
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\Java\agent.jar" DeserializarPersona  
Objeto persona leido: Carlos tiene 30 años.  
Process finished with exit code 0
```

Atributos serializables y no-serializables

Hasta ahora, hemos supuesto que todos los atributos de nuestro objeto **Persona** eran serializables, pero puede ser que no necesitemos almacenar el estado completo del objeto, si no solamente los atributos necesarios. Si alguno no lo es, debemos marcarlo como **transient** para que se ignore. Lo probamos con la **edad**:

```
1 public class Persona implements Serializable {  
2     private String nombre;  
3     private transient int edad;  
4  
5     public Persona(String nombre, int edad) {  
6         this.nombre = nombre;  
7         this.edad = edad;  
8     }  
9  
10    public String getNombre() {  
11        return nombre;  
12    }  
13  
14    public void setNombre(String nombre) {  
15        this.nombre = nombre;  
16    }  
17  
18    public int getEdad() {  
19        return edad;  
20    }  
21  
22    public void setEdad(int edad) {  
23        this.edad = edad;  
24    }  
25  
26    @Override  
27    public String toString() {  
28        return "Persona{" + "nombre=" + nombre + ", edad=" + edad + '}';  
29    }  
30}
```

```

7     this.nombre = nombre;
8     this.edad = edad;
9 }
10
11 public String toString() {
12     return nombre + " tiene " + edad + " años.";
13 }

```

Al deserializar, su valor será *null* (para objetos) o *0*, *false*, etc., según el tipo primitivo que utilice. Vamos a probarlo...

Volvemos a lanzar los métodos para serializar y deserializar personas:

The screenshot shows a dark-themed IDE interface. In the top navigation bar, there are tabs for 'Run' and a selected tab labeled 'DeserializarPersona'. Below the tabs is a toolbar with icons for back, forward, and search. The main area displays a terminal window with the following text:

```

"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:
Objeto persona leido: Carlos tiene 0 años.

Process finished with exit code 0

```

To the left of the terminal, a file tree is visible, showing files like 'datos_personas.txt', 'ejemplo1.txt', 'persona.ser' (which is highlighted), and 'prueba_escribir.txt'. It also shows directories 'test' and 'target', and a '.gitignore' file.

Como no hemos serializado la *edad*, cuando se deserializa, esta se crea con valor *0* (nulo para enteros).

La serialización en Java es un proceso inherentemente inseguro, ya que puede ser utilizada para fines malintencionados, como la inyección de objetos. Por lo tanto, siempre debemos asegurarnos de deserializar sólo los datos de fuentes en las que confiemos y, si es posible, usar alternativas más seguras como JSON (lo veremos más adelante).

Ejercicios sobre serialización



Ejercicio 1

- a) Crea una clase nueva (con el nombre y atributos que quieras).
 - b) Serializa una lista del tipo de la clase que hayas creado `ArrayList<TipoNuevo>` en el fichero `clase_prueba.ser` y luego recupérala (deserializa).
 - c) Al deserializar, recorre la lista y muestra todos sus elementos junto a los valores de cada atributo.
-



Ejercicio 2

- a) Añade un atributo `transient` a tu clase (del tipo que quieras).
 - b) Comprueba que el nuevo atributo no se serializa y que tiene valor `null` al recuperar al objeto.
-



Ejercicio 3

- a) Maneja bien excepciones al deserializar un fichero que no existe o está dañado. Captura `FileNotFoundException` y `IOException`.
 - b) Si falla, se debe crear el archivo con datos por defecto. Usa el formato de la clase que te hayas creado para los ejercicios anteriores.
-



Ejercicio 4

- a) Serializa un `HashMap<String, TipoNuevo>` con varios objetos.
- b) Guárdalo en `mapa.ser`.

c) Recupéralo y muéstraloo ordenado por claves.



Ejercicio 5

a) Crea una jerarquía con clases *Empleado* y *Jefe* (que extiende de *Empleado* con los atributos *nombre* y *salario*). La clase *Jefe* añadirá el atributo *departamento*. Implementa *Serializable* sólo en la superclase.

b) Serializa una lista que contenga empleados y jefes.

c) Recupera la lista e imprime su contenido.

Ficheros JSON.

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. Es fácil de leer, escribir y parsear, tanto para humanos como para máquinas.



Para entender la estructura de estos archivos debemos tener en cuenta dos conceptos:

- **Objetos**: son colecciones no ordenadas de pares de la forma *nombre:valor* separados por comas y puestos entre llaves:

```
1 | {  
2 |   "nombre": "Juan",  
3 |   "edad": 25  
4 | }
```

- **Listas (arrays o vectores)**: representa una lista ordenada de cero o más valores, los cuales pueden ser de cualquier tipo (pero todos los de una misma lista deben ser del mismo tipo). Los valores se separan por comas, y el vector se mete entre corchetes:

```
1 | {  
2 |   "nombre": "Juan",  
3 |   "edad": 25,  
4 |   "lenguajes": ["Java", "Python", "JavaScript"]  
5 | }
```

Sabiendo esto, vamos a crear un *JSON* que contenga un objeto “estudiantes”. A su vez, este objeto contendrá un array (o lista) de varios estudiantes. Usaremos la herramienta <https://jsonformatter.org/> <<https://jsonformatter.org/>>

```
1 {  
2   "estudiantes" : [  
3     ]  
4   }  
5 }
```

Y cada elemento de esta lista, a su vez, será un objeto que contiene varios campos:

```
1 {  
2   "estudiantes" : [  
3     {  
4       },  
5       {  
6         }  
7     ]  
8   }  
9 }  
10 }
```

Por ejemplo:

```
1 {  
2   "estudiantes": [  
3     {  
4       "nombre": "Juan",  
5       "curso": "1DAM"  
6     },  
7     {  
8       "nombre": "María",  
9       "curso": "1DAW"  
10    }  
11  ]  
12 }
```

Ahora, crearemos otra lista distinta que se llame *profesores*:

```
1 {  
2   "estudiantes" : [  
3     {  
4       }  
5     ]  
6   }  
7 }
```

```
4     "nombre": "Juan",
5     "curso": "1DAM"
6   },
7   {
8     "nombre": "María",
9     "curso": "1DAW"
10    }
11  ],
12 "profesores": [
13
14  ]
15 }
```

Dentro contendrá un objeto por cada profesor, pero con una diferencia respecto a los estudiantes: cada profesor imparte más de un módulo, por lo tanto, uno de los campos de cada profesor será a su vez otra lista:

```
1  {
2   "estudiantes" : [
3     {
4       "nombre": "Juan",
5       "curso": "1DAM"
6     },
7     {
8       "nombre": "María",
9       "curso": "1DAW"
10      }
11    ],
12 "profesores": [
13   {
14     "nombre": "Patricia",
15     "modulos" : ["Programación", "Tutoría"]
16   },
17   {
18     "nombre": "Paco",
19     "modulos": ["Sistemas", "Hardware"]
20   }
21 ]
22 }
23 }
```

```

1 {
2   "estudiantes": [
3     {
4       "nombre": "Juan",
5       "curso": "1DAM"
6     },
7     {
8       "nombre": "María",
9       "curso": "1DAW"
10    }
11  ],
12  "profesores": [
13    [
14      {
15        "nombre": "Patricia",
16        "modulos": ["Programación", "Tutoría"]
17      },
18      {
19        "nombre": "Paco",
20        "modulos": ["Sistemas", "Hardware"]
21      }
22    ]
23  ]

```

Ln: 13 Col: 6

RESUMEN:

- Dentro del *JSON* creado tenemos dos objetos: *estudiantes* y *profesores*. Ambos guardan una lista de objetos.
- Por cada estudiante, guardamos su nombre y el curso en el que está.
- Por cada profesor, guardamos su nombre y una lista de los módulos que imparte.

Para comprobar la estructura de nuestro *JSON*, desde la parte derecha de la aplicación *JSON Formatter*, seleccionamos la opción *Vista*, *Árbol* o *Formulario* (elige la que prefieras):

The screenshot shows a Java IDE interface with a navigation bar at the top. The path 'object ► profesores ► 1 ► modulos ►' is displayed. Below this, a tree view of objects is shown:

- object {2}
 - estudiantes [2]
 - 0 {2}
 - nombre : Juan
 - curso : 1DAM
 - 1 {2}
 - nombre : María
 - curso : 1DAW
 - profesores [2]
 - 0 {2}
 - nombre : Patricia
 - modulos [2]
 - 0 : Programación
 - 1 : Tutoría
 - 1 {2}
 - nombre : Paco
 - modulos [2]
 - 0 : Sistemas
 - 1 : Hardware

Librería GSON.

Gson es una librería de *Google* que permite:

- Convertir objetos *Java* a *JSON* (serialización).
- Convertir *JSON* a objetos *Java* (deserialización).



Para poder usarla, debemos añadir la siguiente dependencia al *pom.xml* nuestro proyecto *Maven*:

```
1 <dependency>
2   <groupId>com.google.code.gson</groupId>
3   <artifactId>gson</artifactId>
4   <version>2.10.1</version>
5 </dependency>
```

Además, crearemos la siguiente clase **PersonaJSON** de referencia para manipularla:

```
@Getter
@ToString
public class PersonaJSON {
    private String nombre;
    private int edad;
    private List<String> lenguajes;

    public PersonaJSON() {}

    public PersonaJSON(String nombre, int edad, List<String> lenguajes)
        this.nombre = nombre;
```

```

13         this.edad = edad;
14         this.lenguajes = new ArrayList<>(lenguajes);
15     }
16 }

```

Observa que es necesario crear un constructor vacío para poder usar *Gson*.

De objeto Java a formato *JSON*:

```

1 public class GsonEjemplo {
2     public static void main(String[] args) {
3
4         Gson gson = new Gson();
5
6         // convertir objeto a JSON
7         PersonaJSON persona = new PersonaJSON("Luis", 25, Arrays.asList
8             String json = gson.toJson(persona);
9             System.out.println("JSON: " + json);
10
11     }
12 }

```

```

"C:\Program Files\Java\jdk-23\bin\java.exe" ...
JSON: {"nombre":"Luis", "edad":25, "lenguajes":["Java", "Python"]}
Process finished with exit code 0
|_

```

Con listas:

```

public class GsonEjemplo {
    public static void main(String[] args) {

        Gson gson = new Gson();

        // convertir objeto a JSON
        ArrayList<PersonaJSON> listaPersonas = new ArrayList<>(Arrays.a

```

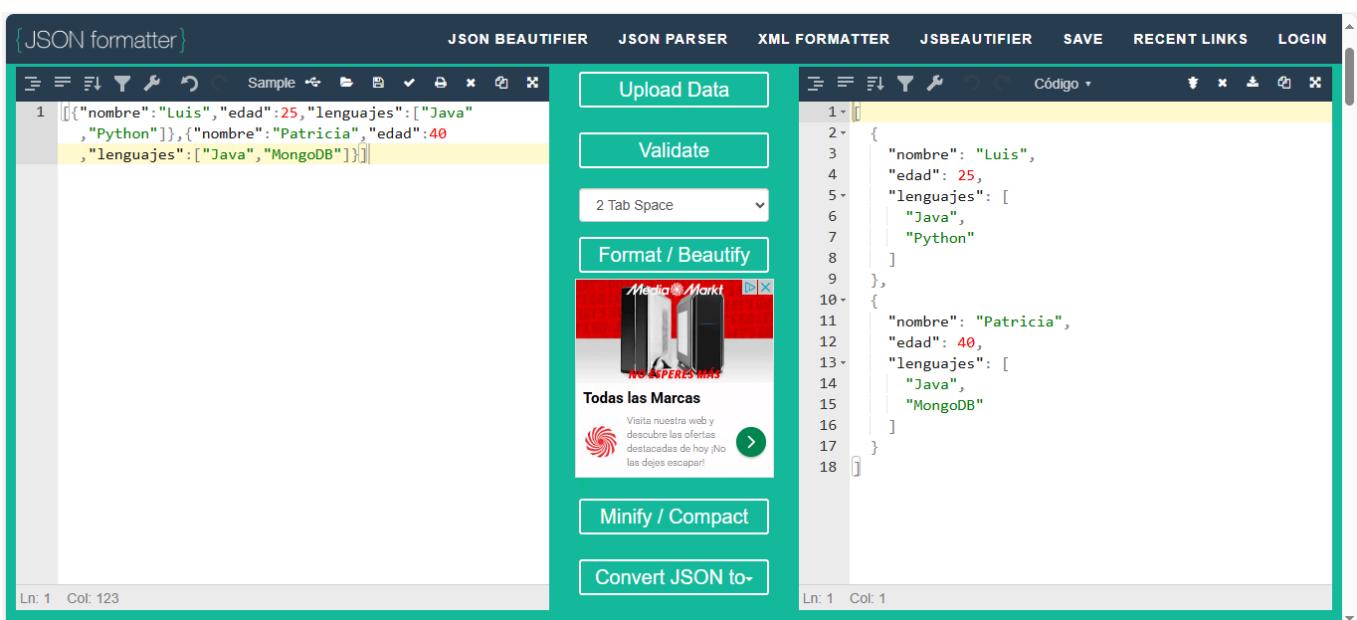
```

8     String json = gson.toJson(listaPersonas);
9     System.out.println("JSON: " + json);
10    }
11 }
12 }
```

```

C:\Program Files\Java\jdk-23\bin\java.exe ...
JSON: [{"nombre": "Luis", "edad": 25, "lenguajes": ["Java", "Python"]}, {"nombre": "Patricia", "edad": 40, "lenguajes": ["Java", "MongoDB"]}]
Process finished with exit code 0
```

Para ver mejor el formato. podemos pedir el *JSON* en alguna herramienta como [Online JSON Formatter <https://jsonformatter.org/>](https://jsonformatter.org/)



o definir el objeto *JSON* como [*GsonBuilder\(\)*](#) desde *Java*:

```

1 | Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

Por ejemplo:

```

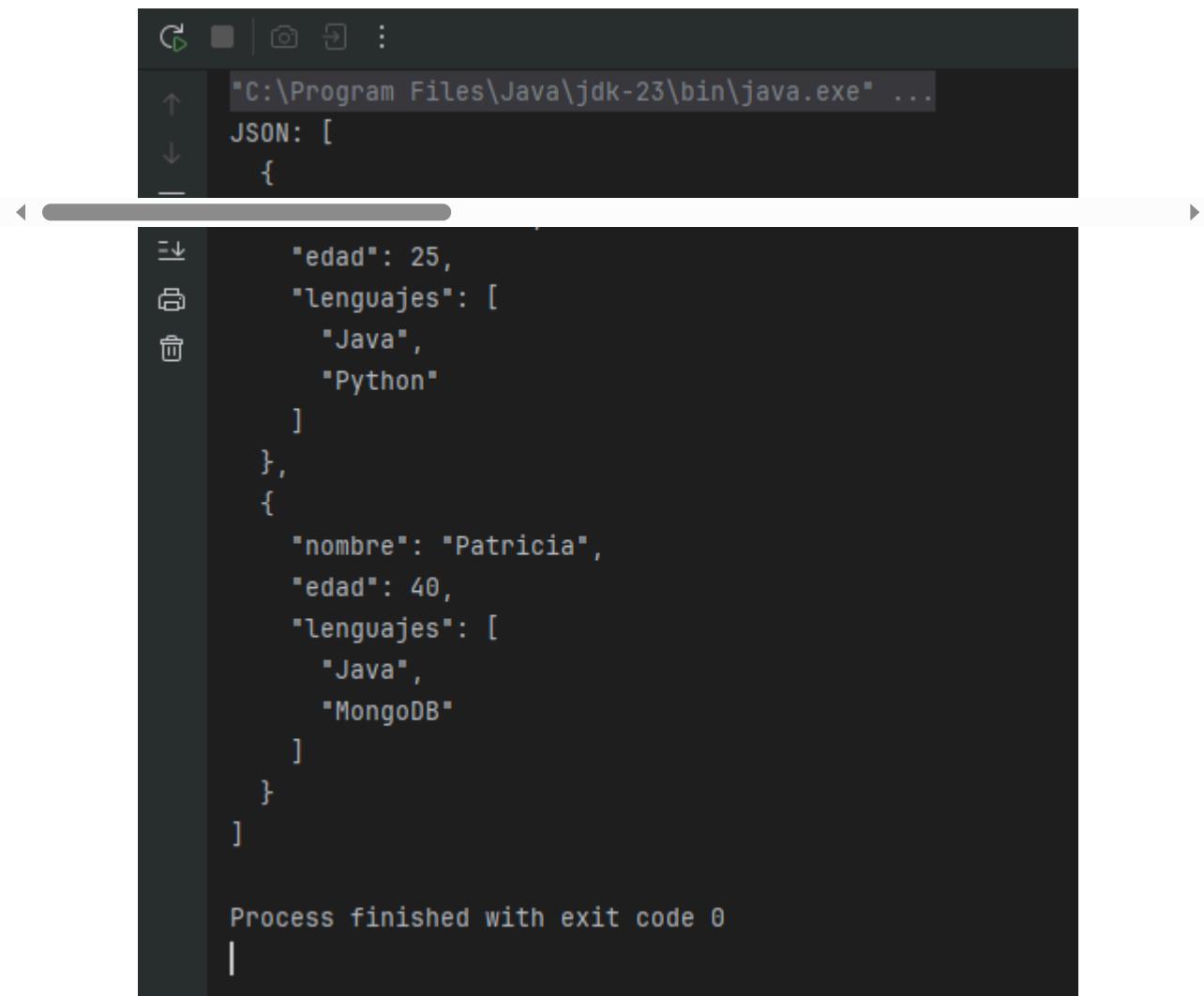
public class GsonEjemplo {
    public static void main(String[] args) {

        Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

```

7         // convertir objeto a JSON
8     ArrayList<PersonaJSON> listaPersonas = new ArrayList<>(Arrays.a
9     String json = gson.toJson(listaPersonas);
10    System.out.println("JSON: " + json);
11
12 }

```



The screenshot shows a terminal window with the following content:

```

C:\Program Files\Java\jdk-23\bin\java.exe" ...
JSON: [
  {
    "edad": 25,
    "lenguajes": [
      "Java",
      "Python"
    ],
    "nombre": "Patricia",
    "edad": 40,
    "lenguajes": [
      "Java",
      "MongoDB"
    ]
  }
]

Process finished with exit code 0
|
```

Guardar el *JSON* en un archivo

Haremos uso de nuestra vieja conocida *FileWriter*:

```

public class GsonEjemplo {
    public static void main(String[] args) {

        Gson gson = new GsonBuilder().setPrettyPrinting().create();

        // convertir objeto a JSON

```

```

7     ArrayList<PersonaJSON> listaPersonas = new ArrayList<>(Arrays.a
8     String json = gson.toJson(listaPersonas);
9     System.out.println("JSON: " + json);
10
11    try (FileWriter writer = new FileWriter("src/main/resources/per
12        gson.toJson(listaPersonas, writer);
13        System.out.println("JSON guardado en persona.json");
14    } catch (Exception e) {
15        System.out.println("Algo ha ido mal.");
16        e.printStackTrace();
17    }
18
19}
20

```

The screenshot shows the Java code running in an IDE. The code reads a list of persons from a file, converts it to JSON, prints the JSON string, and then writes it to a file named 'persona.json'. The output window shows the JSON data and the confirmation message 'JSON guardado en persona.json'.

```

Project Main.java
Main.java
1 [
2 {
3     "nombre": "nombre"
4     "edad": 25,
5     "lenguajes": [
6         "Java",
7         "Python"
8     ]
9 },
10 {
11     "nombre": "Patricia",
12     "edad": 40,
13     "lenguajes": [
14         "Java",
15         "MongoDB"
16     ]
17 }
18 ]
19 JSON guardado en persona.json
20
Process finished with exit code 0

```

```
1 [  
2 {  
3     "nombre": "Luis",  
4     "edad": 25,  
5     "lenguajes": [  
6         "Java",  
7         "Python"  
8     ]  
9 },  
10 {  
11     "nombre": "Patricia",  
12     "edad": 40,  
13     "lenguajes": [  
14         "Java",  
15         "MongoDB"  
16     ]  
17 }  
18 ]
```

De formato *JSON* a objeto *Java*:

```
1 public class GsonEjemplo {  
2     public static void main(String[] args) {  
3  
4         Gson gson = new Gson();  
5  
6         // convertir objeto a JSON  
7         PersonaJSON persona = new PersonaJSON("Luis", 25, Arrays.asList  
8             String json = gson.toJson(persona);  
9             System.out.println("JSON: " + json);  
10  
11         // convertir JSON a objeto  
12         PersonaJSON persona2 = gson.fromJson(json, PersonaJSON.class);  
13         System.out.println("Nombre: " + persona2.getNombre() + ", Edad:  
14     }  
15 }
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
JSON: {"nombre":"Luis","edad":25,"lenguajes":["Java","Python"]}
Nombre: Luis, Edad: 25 Lenguajes: [Java, Python]

Process finished with exit code 0
```

Leer el *JSON* desde el archivo

Usaremos la clase *FileReader*, que también conocemos de ejercicios previos.

```
public class GsonEjemplo {
    public static void main(String[] args) {

        Gson gson = new GsonBuilder().setPrettyPrinting().create();

        // convertir objeto a fichero JSON
        PersonaJSON persona = new PersonaJSON("Luis", 25, Arrays.asList(
            String json = gson.toJson(persona);
            System.out.println("JSON: " + json);

        try (FileWriter writer = new FileWriter("src/main/resources/persona.json")) {
            gson.toJson(persona, writer);
            System.out.println("JSON guardado en persona.json");
        } catch (Exception e) {
            System.out.println("Algo ha ido mal.");
            e.printStackTrace();
        }

        // convertir fichero JSON a objeto
        try {
            FileReader reader = new FileReader("src/main/resources/persona.json");
            PersonaJSON persona2 = gson.fromJson(reader, PersonaJSON.class);
            System.out.println("Nombre: " + persona2.getNombre());
            System.out.println("Edad: " + persona2.getEdad());
            System.out.println("Lenguajes: " + persona2.getLenguajes());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
        }  
    }  
}
```

The terminal window shows the following output:

```
C:\Program Files\Java\jdk-23\bin\java.exe" ...  
JSON: {  
    "nombre": "Luis",  
    "edad": 25,  
    "lenguajes": [  
        "Java",  
        "Python"  
    ]  
}  
JSON guardado en persona.json  
Nombre: Luis  
Edad: 25  
Lenguajes: [Java, Python]  
  
Process finished with exit code 0
```

The code editor shows the following JSON code:

```
1 {  
2     "nombre": "Luis",  
3     "edad": 25,  
4     "lenguajes": [  
5         "Java",  
6         "Python"  
7     ]  
8 }
```



Práctica (1)

Crea un programa en *Java* que gestione el inventario de una tienda de videojuegos.

Cada videojuego debe tener:

- Nombre del juego.

- Plataforma (por ejemplo: PC, PS5, Xbox).
- Precio.
- Disponible (booleano).
- Una lista de géneros (acción, estrategia, etc.).

El programa debe:

- a) Crear 3 videojuegos distintos **por consola** y guardarlos en una colección.
 - b) Guardar toda la colección en un archivo *JSON* (*videojuegos.json*).
 - c) Leer ese archivo y mostrarlo por pantalla.
 - d) Reconstruir la colección de objetos *Java* a partir del archivo guardado.
 - e) Añadir un videojuego nuevo a la colección.
 - f) Mostrar en consola los videojuegos cuyo precio sea menor a 30€.
 - g) Volver a guardar la lista actualizada en el archivo *JSON*.
-

BONUS. Consumiendo APIs externas desde Java.



API significa *Application Programming Interface*, o en español, **Interfaz de Programación de Aplicaciones**. Es una forma estandarizada de que dos programas o servicios se comuniquen entre sí.

Una API define cómo pedir datos o enviar instrucciones a otro programa o servicio, sin tener que saber cómo funciona por dentro (caja negra). Los datos que se intercambian son principalmente *JSON*, que como ya hemos comentado, es un formato de texto ligero y bastante descriptivo.

Tipos comunes de APIs

- **REST APIs:** usan HTTP (como páginas web) y JSON. Son las más comunes, y las que veremos nosotros.
- **SOAP APIs:** más antiguas, basadas en XML.
- **WebSockets/APIs en tiempo real:** para juegos, chat, etc.

Uso de *Postman* para explorar APIs

Postman es una herramienta para probar *APIs REST*, ver respuestas *JSON* y explorar *endpoints* en general.



POSTMAN

Download Postman | Get Started for Free <<https://www.postman.com/downloads/>>

Crearemos una nueva *Colección y petición*:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Team Workspace' selected, showing 'Collections', 'Environments', 'Flows', and 'History'. In the main area, a 'New Collection' dialog is open. It has tabs for 'Overview', 'Authorization', 'Scripts', 'Tests', 'Variables', and 'Runs'. The 'Overview' tab is active. Below it, there are three templates: 'REST API basics', 'End-to-end testing', and 'Functional testing'. To the right, there are statistics: 0 requests, 0 forks, 0 watchers, and a note that it was created by 'You'. At the bottom, there's a section for 'Recent changes' with a log entry for 'May 19, 2025' at '10:47 AM' by 'Patricia'.

Por defecto, nos aparece una petición de tipo **GET**, pero podríamos usar cualquiera de las que hay disponibles:

The screenshot shows a specific request configuration in Postman. The URL field contains 'https://v2.jokeapi.dev/joke/Programming?type=single&lang=es'. The method dropdown is set to 'GET'. The request body is empty. The 'Headers' section shows six entries. The 'Send' button is visible at the top right.

Usaremos GET con la siguiente URL: <https://v2.jokeapi.dev/joke/Programming?type=single&lang=es>

HTTP New Collection / Chistes

GET https://v2.jokeapi.dev/joke/Programming?type=single&lang=es

Params • Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
type	single		
lang	es		
Key	Value	Description	

Se trata de una API que devuelve chistes sobre Programación. Probamos a lanzarla:

HTTP New Collection / Chistes

GET https://v2.jokeapi.dev/joke/Programming?type=single&lang=es

Params • Authorization Headers (6) Body Scripts Tests Settings Cookies

Body Cookies Headers (29) Test Results | ⚙️ 200 OK • 180 ms • 1.72 KB • 🗑️ Save Response ⚙️

{ } JSON ▾ ▶ Preview ⚡ Visualize | ⚙️

```

1  {
2      "error": false,
3      "category": "Programming",
4      "type": "single",
5      "joke": "No te despedirán del trabajo, si nunca comentas tu código y además eres el único que sabe cómo
6          funciona",
7      "flags": {
8          "nsfw": false,
9          "religious": false,
10         "political": false,
11         "racist": false,
12         "sexist": false,
13         "explicit": false
14     },
15     "safe": true,
16     "id": 3,
17     "lang": "es"
18 }
```

Si devuelve un **estatus 200 OK**, es que nuestra petición ha sido correcta.

```
{
    "error": false,
    "category": "Programming",
    "type": "single",
    "joke": "No te despedirán del trabajo, si nunca comentas tu código :)",
    "flags": {
        "nsfw": false,
        "religious": false,
        "political": false,
```

```

11     "racist": false,
12     "sexist": false,
13     "explicit": false
14   },
15   "safe": true,
16   "id": 3,
17   "lang": "es"
}

```

Para poder invocar APIs, es muy importante saber qué parámetros acepta el *endpoint*. En nuestro caso, le hemos pedido chistes en Español (lang=es). Prueba a mandar un idioma inventado y observa lo que pasa:

The screenshot shows a Postman collection named 'Chistes'. A GET request is being made to <https://v2.jokeapi.dev/joke/Programming?type=single&lang=españístico>. The 'Params' tab is selected, showing three parameters: 'Key' (checked), 'Value' (single), 'Description' (empty), and 'Bulk Edit' (checkbox). Another 'Key' (checked) is listed but has no value. The 'Body' tab shows an empty JSON object. The response status is 400 Bad Request, with a timestamp of 1747644965148.

```

1 {
2   "error": true,
3   "internalError": false,
4   "code": 106,
5   "message": "No matching joke found",
6   "causedBy": [
7     "No jokes were found that match your provided filter(s)."
8   ],
9   "additionalInfo": "The specified language code \\\"españístico\\\" is invalid. Please see https://jokeapi.dev/#langcodes-endpoint for more info."
10 }
11

```

Nos devuelve un status **400 bad request**, por lo que saber cómo interactuar con las APIs es muy importante. Para eso, deberemos consultar la información de cada API en su documentación de referencia. Por ejemplo, la de *jokeapi*. **JokeAPI - Documentation <<https://sv443.net/jokeapi/v2/>>**

Select category / categories: Any Custom: Programming Misc Dark Pun Spooky Christmas

Select language: en - English

Select flags to blacklist: (optional) nsfw religious political racist sexist explicit

Select response format: default (json) xml yaml plain text

Select at least one joke type: single twopart

Search for a joke that contains this search string: (optional)

Search for a joke in this ID range: (optional) From: 0 To: 1367

Amount of jokes: 1

Vale, ya lo he visto en Postman, pero... ¿cómo usamos una API desde Java?

1. Creamos la clase que represente la estructura del JSON.

Usaremos la herramienta **JSON to POJO Object Online Converter - Json2CSharp Toolkit** <<https://json2csharp.com/code-converters/json-to-pojo>>

The screenshot shows the JSON2CSharp website interface. At the top, there's a banner for AirEuropa with a Panama flag and the text "Más de lo que imaginas". Below the banner, the main title is "Convert JSON to POJO Objects in Java Online". The JSON input is:

```

1 {
  "error": false,
  "category": "Programming",
  "type": "single",
  "joke": "Sólo hay 10 tipos de personas en este mundo.",
  "flags": {
    "nsfw": false,
    "religious": false,
    "political": false,
    "racist": false,
    "sexist": false,
    "explicit": false
  },
  "safe": true,
  "id": 5,
  "lang": "es"
}

```

The generated Java code is:

```

public class Chistes {
    public boolean error;
    public String category;
    public String type;
    public String joke;
    public Flags flags;
    public boolean safe;
    public int id;
    public String lang;
}

public class Flags {
    public boolean religious;
    public boolean political;
    public boolean racist;
    public boolean sexist;
    public boolean explicit;
}

public class Root{
    public boolean error;
    public String category;
    public String type;
    public String joke;
    public Flags flags;
    public boolean safe;
    public int id;
    public String lang;
}

```

At the bottom, there are buttons for "Use Properties", "Convert", and "Copy To Clipboard".

¿Y si hay campos que no me interesan? Puedes ignorarlos, *Gson* no se quejará si hay campos extra. Nosotros, eliminaremos el atributo *Flags*.

```

1 public class Chistes {
2
3     public boolean error;
4     public String category;
5     public String type;

```

```
6     public String joke;
7     public boolean safe;
8     public int id;
9     public String lang;
10    }
11 }
```

2. Hacemos una solicitud HTTP (con *HttpURLConnection*):

```
1     String apiUrl = "https://v2.jokeapi.dev/joke/Programming?ty
2
3     URL url = new URL(apiUrl);
4     HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
5     conexion.setRequestMethod("GET");
```

3. Recibimos una respuesta en *JSON*.

```
1     BufferedReader in = new BufferedReader(new InputStreamReader(
2         new URL(apiUrl).openStream()));
3     String json = "";
4     String line;
5     while ((line = in.readLine()) != null) {
6         json.append(line);
7     }
8     in.close();
```

4. Usamos una librería como *Gson* para convertir ese *JSON* en objetos *Java*.

```
1     Gson gson = new Gson();
2     Chistes chiste = gson.fromJson(json.toString(), Chistes.class);
```

5. Mostramos o procesamos los datos.

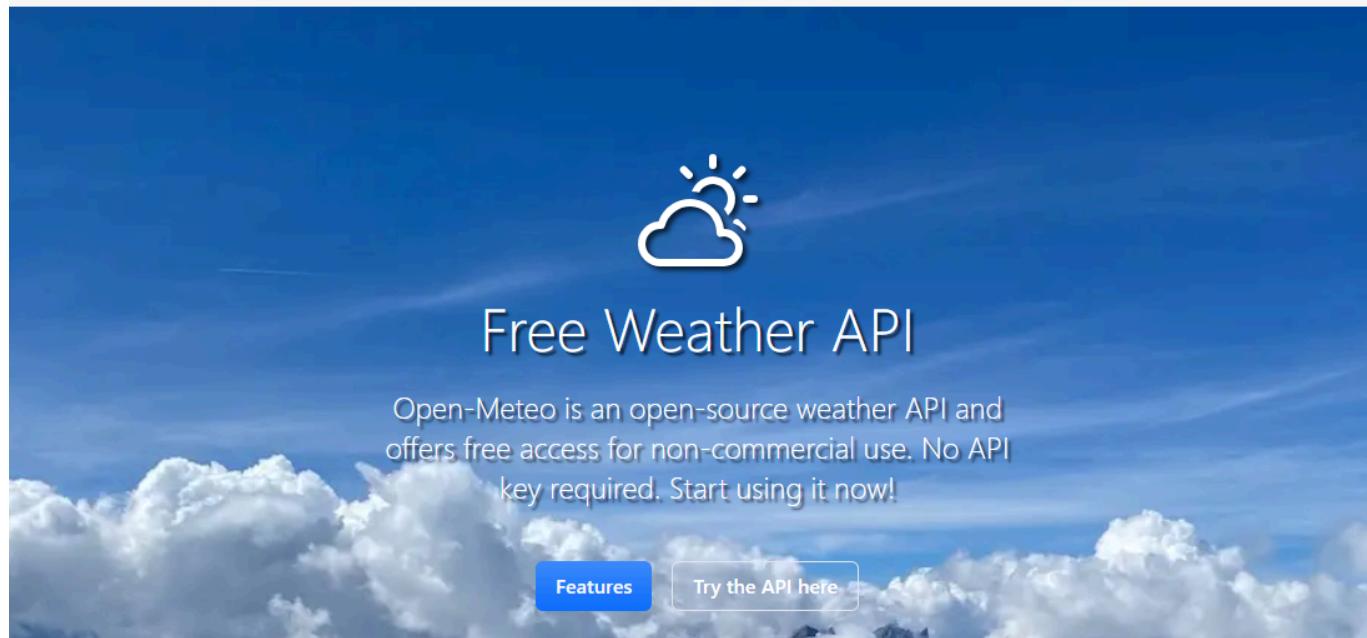
```
1     System.out.println("Chiste:");
2     System.out.println(chiste.joke);
```

Programa completo:

```
1 public class APIchistes {
2
3     public static void main(String[] args) {
4
5         try {
6
7             String apiUrl = "https://v2.jokeapi.dev/joke/Programming?ty
8
9             URL url = new URL(apiUrl);
10            HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
11            conexion.setRequestMethod("GET");
12
13            BufferedReader in = new BufferedReader(new InputStreamReader(
14                new StringBuilder());
15            String line;
16            while ((line = in.readLine()) != null) {
17                json.append(line);
18            }
19            in.close();
20
21            Gson gson = new Gson();
22            Chistes chiste = gson.fromJson(json.toString(), Chistes.class);
23
24            System.out.println("Chiste:");
25            System.out.println(chiste.joke);
26
27        } catch (Exception e) {
28            System.out.println("Algo ha ido mal.");
29            e.printStackTrace();
30        }
31    }
32}
33}
```

Otro ejemplo: API del clima

Open-Meteo <<https://open-meteo.com/>> es una API gratuita de pronóstico del tiempo, sin autenticación.



Ejemplo de llamada:

```
1 | https://api.open-meteo.com/v1/forecast?latitude=38.41921088667074&longitude=-0.4482154992747586
```

Mutxamel: 38.41974886121347, -0.4482154992747586

Puedes consultar toda su info aquí: [Docs](#) | [Open-Meteo.com](https://open-meteo.com/en/docs) <<https://open-meteo.com/en/docs>>

Vamos a *Postman* para probar el *endpoint*:

HTTP New Collection / Chistes

Save Share

GET https://api.open-meteo.com/v1/forecast?latitude=38.41921088667074&longitude=-0.4464988854601825¤t_weather=true

Send

Params • Authorization Headers (6) Body Scripts Tests Settings Cookies

Body Cookies Headers (5) Test Results | ⚙️ 200 OK • 281 ms • 456 B • ⚙️ Save Response ⚙️

{ } JSON ▾ ▶ Preview ⚙️ Visualize | ▾

```

1  {
2      "latitude": 38.4375,
3      "longitude": -0.4375,
4      "generationtime_ms": 0.0400543212890625,
5      "utc_offset_seconds": 0,
6      "timezone": "GMT",
7      "timezone_abbreviation": "GMT",
8      "elevation": 77.0,
9      "current_weather_units": {
10         "time": "iso8601",
11         "interval": "seconds",
12         "temperature": "°C",
13         "windspeed": "km/h",
14         "winddirection": "°",
15         "is_day": "",
16         "weathercode": "wmo code"
17     },
18     "current_weather": {
19         "time": "2025-05-19T09:30",
20         "interval": 900,
21         "temperature": 23.4,
22         "windspeed": 4.7,

```

Como funciona, le damos el JSON de retorno a json2csharp.com para que nos cree las clases en Java:

JSON2CSharp

Blog All Tools

PANAMA
Más de lo que imaginas
Vuela desde 308 EUR por trayecto Compra ya

Convert JSON to POJO Objects in Java Online

Convert any JSON object to a POJO JAVA class online. Check below panel on how to use this converter and how to deserialize using Jackson library.

JSON → JAVA JSON → C XML → C XML → JAVA JSON → Python JSON → Dart JSON → Gass

```

1  {
2      "latitude": 38.4375,
3      "longitude": -0.4375,
4      "generationtime_ms": 0.0400543212890625,
5      "utc_offset_seconds": 0,
6      "timezone": "GMT",
7      "timezone_abbreviation": "GMT",
8      "elevation": 77.0,
9      "current_weather_units": {
10         "time": "iso8601",
11         "interval": "seconds",
12         "temperature": "°C",
13         "windspeed": "km/h",
14         "winddirection": "°",
15         "is_day": "",
16         "weathercode": "wmo code"
17     },
18     "current_weather": {
19         "time": "2025-05-19T09:30",
20         "interval": 900,

```

// import com.fasterxml.jackson.databind.ObjectMapper
/* ObjectMapper om = new ObjectMapper();
Root root = om.readValue(myJsonString, Root.class);
public class CurrentWeather{
 public String time;
 public int interval;
 public double temperature;
 public double windspeed;
 public int winddirection;
 public int is_day;
 public int weathercode;
}
public class CurrentWeatherUnits{
 public String time;
 public String interval;
 public String temperature;
 public String windspeed;

Use Properties Convert Copy To Clipboard

Sólo vamos a quedarnos con la clase **CurrentWeather**.

```

1  public class CurrentWeather{
2      public String time;
3

```

```
4     public int interval;
5     public double temperature;
6     public double windspeed;
7     public int winddirection;
8     public int is_day;
9     public int weathercode;
10    }
```

Pero como vamos a obtener la estructura de la clase *Root* (para nosotros, se llamará *ElTiempo*), crearemos una clase estática dentro de ella para simplificar la estructura:

```
1  public class ElTiempo {
2      public double latitude;
3      public double longitude;
4      public double generationtime_ms;
5      public int utc_offset_seconds;
6      public String timezone;
7      public String timezone_abbreviation;
8      public double elevation;
9      CurrentWeather current_weather;
10
11     static class CurrentWeather{
12         public String time;
13         public int interval;
14         public double temperature;
15         public double windspeed;
16         public int winddirection;
17         public int is_day;
18         public int weathercode;
19
20     }
21 }
```

Ahora, seguiremos el mismo patrón del programa principal usado para los chistes:

```
public class Clima {
    public static void main(String[] args) {
        try {

            String apiUrl = "https://api.open-meteo.com/v1/forecast?lat=
URL url = new URL(apiUrl);
```

```

8     HttpURLConnection con = (HttpURLConnection) url.openConnection();
9     con.setRequestMethod("GET");
10
11    BufferedReader in = new BufferedReader(new InputStreamReader(
12        response.toString().getBytes("UTF-8")));
13    String line;
14    while ((line = in.readLine()) != null) {
15        response.append(line);
16    }
17    in.close();
18
19    Gson gson = new Gson();
20    ElTiempo weather = gson.fromJson(response.toString(), ElTiempo.class);
21
22    System.out.println("Clima actual en Mutxamel:");
23    System.out.println("Temperatura: " + weather.current_weather.temperature);
24    System.out.println("Viento: " + weather.current_weather.wind_speed);
25    System.out.println("Hora del dato: " + weather.current_weather.last_update);
26
27} catch (Exception e) {
28    e.printStackTrace();
29}
30}
31}

```

```

"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Clima actual en Mutxamel:
Temperatura: 23.0°C
Viento: 7.6 km/h
Process finished with exit code 0

```



Práctica (2)

Replica los pasos anteriores para consumir la API que tú quieras. Si no se te ocurre cuál, usa alguna de estas:

- **CoinGecko API – Criptomonedas.** Consulta precios de *Bitcoin*, *Ethereum*, etc. en tiempo real.

URL ejemplo:

[<https://api.coingecko.com/api/v3/simple/price?
ids=bitcoin,ethereum&vs_currencies=usd,eur>](https://api.coingecko.com/api/v3/simple/price?ids=bitcoin,ethereum&vs_currencies=usd,eur)

- **Open Trivia DB.** Preguntas de trivial por categoría, dificultad, etc.

URL ejemplo:

[<https://opentdb.com/api.php?amount=1&category=18>](https://opentdb.com/api.php?amount=1&category=18)

Webs para encontrar APIs:

- **RapidAPI** <<https://rapidapi.com/hub>>
- <https://public-apis.io/> <<https://public-apis.io/>>
- GitHub - public-apis/public-apis: A collective list of free APIs
<<https://github.com/public-apis/public-apis>>
- <https://apilist.fun/> <<https://apilist.fun/>>



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<<http://creativecommons.org/licenses/by-sa/4.0/>>

