Calculadora en JavaFX (IntelliJ)

1. Introducción

Vamos a intentar juntar todo lo aprendido en una guía para realizar una aplicación `JavaFX` con `SceneBuilder` i `IntelliJ`, siguiendo el modelo `MVC`.

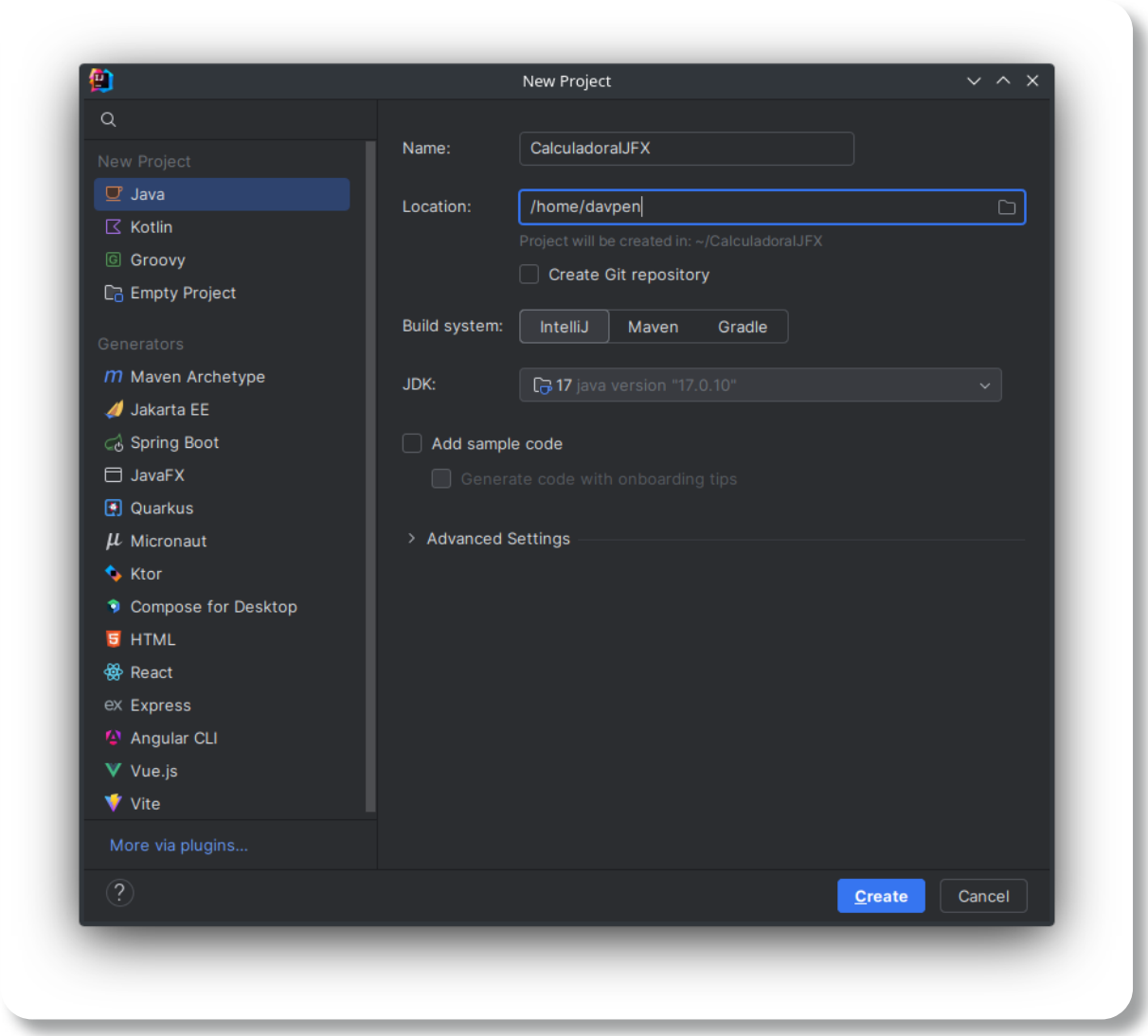
Necesitaras:

- IntelliJ Ultimate (seguramente con la comunity también funcione)
- OpenJDK 17 (seguramente funcionará con una posterior)
- JavaFx 21 LTS
- SceneBuilder

2. Crear proyecto

Vamos a crear el proyecto como siempre `Java` no necesitamos el asistente de `JavaFX`.

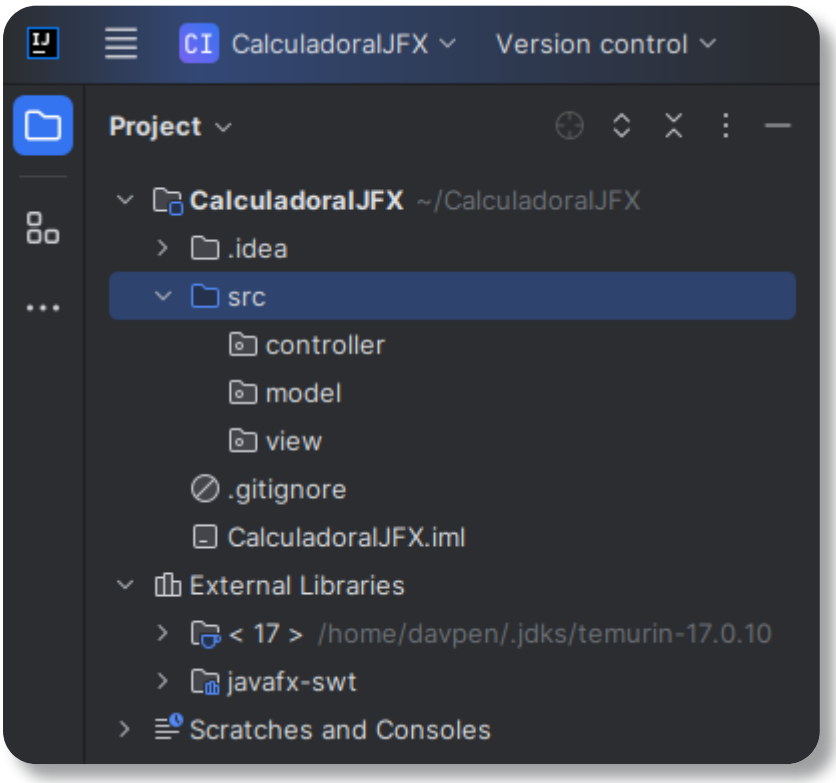
En nuestro caso llamamos al proyecto `CalculadoraIJFX` y desmarcamos la opción de `Add sample code`.



Ahora para seguir el modelo **MVC** crearemos los tres *packages* (`controller` , `model` y `view`).

Luego añadimos al proyecto la librería de `JavaFX21` como hemos visto en otros documentos de la unidad.

En este momento nuestro proyecto debería tener este aspecto:



Añadimos las propiedades correctas a las `VM options` del proyecto. Tal y como hemos visto en otros documentos de la unidad. `Menu de 4 rallas`, `Run`, `Edit Configurations...`, `Edit configuration templates...`, `Add VM options`:

> **Importante** La última parte `,javafx.fxml` solo es necesaria en el caso que usemos formularios `fxml`, por eso no lo habíamos puesto hasta ahora.

```
1 --module-path /home/davpen/Nextcloud/DOCENCIA/PRG-2324/javafx-sdk-21.0.3/lib --add-
  modules=javafx.controls,javafx.fxml
```

3. Modelo

Para la calculadora necesitaremos un modelo que se encargue de realizar las distintas operaciones de nuestra calculadora. Para ello crearemos un nuevo fichero `Operaciones.java` dentro del paquete `model` con el siguiente contenido:

```java
package model;

public class Operaciones {
    private double operador1;
    private double operador2;

    public Operaciones(double operador1, double operador2) {
        this.operador1 = operador1;
        this.operador2 = operador2;
    }

    public double getOperador1() {
        return operador1;
    }

    public void setOperador1(double operador1) {
        this.operador1 = operador1;
    }

    public double getOperador2() {
        return operador2;
    }

    public void setOperador2(double operador2) {
        this.operador2 = operador2;
    }

    public double suma(){
        return this.operador1+this.operador2;
    }
    public double resta(){
        return this.operador1-this.operador2;
    }
    public double multiplicacion(){
        return this.operador1*this.operador2;
    }
    public double division(){
        return this.operador1/this.operador2;
    }
}
```

Fíjate que este es un modelo muy simple, con dos atributos, un constructor, sus *getters* y *setters* y las cuatro operaciones básicas de nuestra calculadora (sumar, restar, multiplicar y dividir).

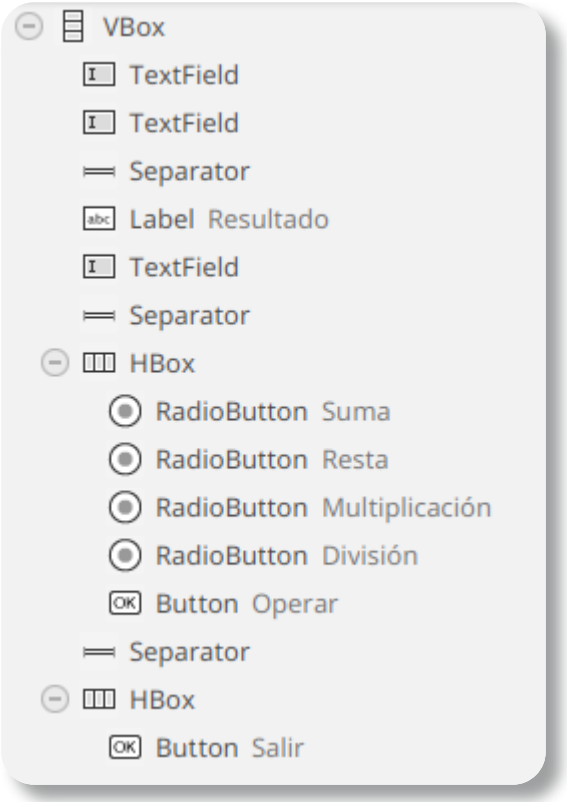
4. Vista

Ahora vamos añadir un nuevo fichero `FXML File` al paquete `view` (Si no te aparece la opción es que no tienes bien configurado el proyecto de IntelliJ). Llamaremos `Calculadora.fxml` al nuevo fichero.

Ahora pulsamos el botón derecho sobre el nuevo fichero y elegimos la opción `Open in SceneBuilder`, nos preguntará donde tenemos instalado el SceneBuilder, se lo indicamos (en linux, por ejemplo, está en (`/opt/scenebuilder/bin/SceneBuilder`) y nos mostrará el SceneBuilder con el formulario Calculadora vacio.

Ahora deberías crear una ventana similar a esta:

Este ejemplo tiene la siguiente jerarquía:



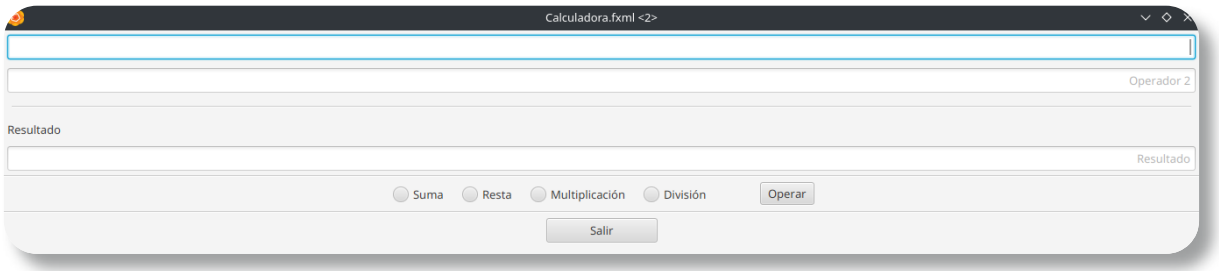
Recuerda dar nombre a todos los componentes en la pestaña `code` al campo `fx:id`.

`txtOperador1`, `txtOperador2` y `txtResultado` para los `TextField's`

`rbSuma`, `rbResta`, `rbMultiplicación`, `rbDivision` para los `RadioButton's`

Desactiva el `txtResultado`, para que no sea editable.

Crea los contenedores y ajusta sus alineaciones, así como los márgenes y espaciadores de los elementos que contienen, de manera que si amplias la ventana al máximo quede algo similar a esto:



También debes añadir la acciones `ON ACTION` dentro de la pestaña `code` para los botones:

`btnSalir` : # `salir`

`btnOperar` : # `operar`

5. Controladores

Necesitaremos dos archivos dentro del package de `controller`:

1. El controlador para la vista de la calculadora.
2. La clase main que cargará la vista principal

5.1. `CalculadoraController.java`

Realizar el controlador para la vista es muy sencillo y automático (Si dispones del plugin `FXMLManager`).

Añade el texto `fx:controller="view.CalculadoraController"` al final de la linea del VBOX del archivo `Calculadora.fxml`:

```
<VBox maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="253.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/19"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="controller.CalculadoraController">
```

Una vez configurado todo esto debes hacer clic derecho sobre el archivo `Calculadora.fxml` y elegir la opción "`Update Controller from FXML`" (Esto deberás hacerlo cada vez que realices un cambio en el fichero `FXML`).

Ahora, dentro del `CalculadoraController.java` agregaremos el código necesario para gestionar las acciones de los botones, y además asegurarnos que los *radio buttons* son auto-excluyentes:

Acción `salir`:

```
    @javafx.fxml.FXML
    private void salir(ActionEvent event) {
        Stage stage = (Stage) btnSalir.getScene().getWindow();
        stage.close();
    }
```

Acción `operar`:

```
    @javafx.fxml.FXML
    private void operar(ActionEvent event) {
        try {
            double op1 = Double.parseDouble(this.txtOperador1.getText());
            double op2 = Double.parseDouble(this.txtOperador2.getText());
            Operaciones op = new Operaciones(op1, op2);
            if (this.rbSuma.isSelected()) {
                this.txtResultado.setText(String.valueOf(op.suma()));
            } else if (this.rbResta.isSelected()) {
                this.txtResultado.setText(String.valueOf(op.resta()));
            } else if (this.rbMultiplicacion.isSelected()) {
                this.txtResultado.setText(String.valueOf(op.multiplicacion()));
            } else if (this.rbDivision.isSelected()) {
                if (op2 != 0) {
                    this.txtResultado.setText(String.valueOf(op.division()));
                } else {
                    Alert alert = new Alert(Alert.AlertType.ERROR);
                    alert.setHeaderText(null);
                    alert.setTitle("Error");
                    alert.setContentText("El operador 2 no puede ser 0.");
                    alert.showAndWait();
                }
            }
        } catch (NumberFormatException numberFormatException) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setHeaderText(null);
            alert.setTitle("Error");
            alert.setContentText("Formato incorrecto de algun operando");
            alert.showAndWait();
        }
    }
```

Recuerda realizar el *import* del `model.Operaciones` y de todos los componentes de javaFx necesarios:

```
import model.Operaciones;
import javafx.event.ActionEvent;
import javafx.scene.control.*;
import javafx.stage.Stage;
```

Acción `initialize`:

```
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        ToggleGroup tgRadio = new ToggleGroup();
        rbSuma.setToggleGroup(tgRadio);
        rbMultiplicacion.setToggleGroup(tgRadio);
        rbResta.setToggleGroup(tgRadio);
        rbDivision.setToggleGroup(tgRadio);
    }
```

El método `initialize` será llamado al instanciar el controlador y generará un `ToggleGroup` de manera que solo podamos seleccionar una de las cuatro opciones disponibles.

5.2. `main.java`

Por último solo nos queda añadir la clase `main`, que contendrá el método `main` que lanzará la aplicación JavaFX.

Para ello en el paquete `controller` pulsamos botón derecho y añadimos un fichero de tipo `JavaFX Application` y le llamaremos `Main.java`.

IntelliJ genera un método `start` de ejemplo, que nostotros sustituiremos por el siguiente código para que cargue nuestra vista:

```
    @Override
    public void start(Stage primaryStage) {
        try {
            Parent root =
FXMLLoader.load(Objects.requireNonNull(ClassLoader.getSystemClassLoader().getResource("view
/Calculadora.fxml")));
            Scene scene = new Scene(root);
            primaryStage.setTitle("CalculadoraIJFX");
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
```

Necesitaremos algunos imports:

```
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
```

Ampliación **Different ways to load classpath resources in Java**

A comparison of different ways of resources loading in Java

Followings are the preferred ways to load resources in classpath.

- `this.getClass().getResource(resourceName)`: It tries to find the resource in the same package as 'this' class unless we use absolute path starting with '/'
- `Thread.currentThread().getContextClassLoader().getResource(resourceName)`: A ClassLoader can be passed (shared) when creating a new thread using `Thread.setContextClassLoader`, so that different thread contexts can load each other classes/resources. If not set, the default is the ClassLoader context of the parent Thread. This

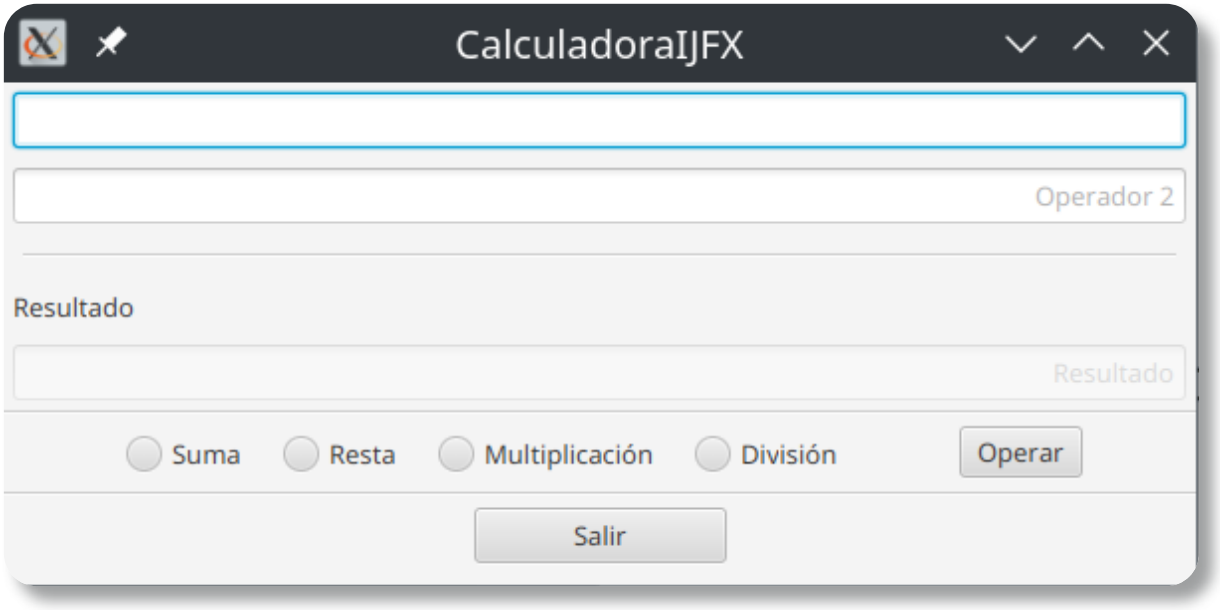
method is not appropriate if we want to load resources inside the packages unless we use complete paths starting from root.

- `ClassLoader.getSystemClassLoader().getResource(resourceName)` : `ClassLoader.getSystemClassLoader()` gives the class loader used to start the application. we have to use complete path for the resources starting from root.

If we don't create any threads in the entire application, the main thread will end up with the system class loader as their context class loader.

6. Primer lanzamiento

Si todo ha ido bien debería aparecer nuestra calculadora en pantalla:



7. Tarea Aules

La tarea consiste en seguir la guia y crear tu aplicación en IntelliJ, pon como título de aplicación tu nombre y apellidos y a continuación genera el archivo jar siguiendo estas instrucciones:

Instructions:

- `File` -> `Project Structure` -> `Project Settings` -> `Artifacts` -> Click `+` (plus sign) -> `Jar` -> `From modules with dependencies...`
- Select a Main Class (the one with `main()` method) if you need to make the jar runnable.
- Select `Extract to the target Jar`
- Click `OK`
- Click `Apply/OK`

The above sets the "skeleton" to where the jar will be saved to. To actually build and save it do the following:

- `Build` -> `Build Artifact` -> `Build`

Try Extracting the .jar file from:

```
📦ProjectName
 ┗ 📂out
   ┗ 📂artifacts
     ┗ 📂ProjectName_jar
       ┗ 📜ProjectName.jar
```

References:

- (Aug 2010) http://blogs.jetbrains.com/idea/2010/08/quickly-create-jar-artifact/ (Here's how to build a jar with IntelliJ 10)
- (Mar 2023) https://www.jetbrains.com/help/idea/compiling-applications.html#package_into_jar

Extraido de: https://stackoverflow.com/questions/1082580/how-to-build-jars-from-intellij-idea-properly

Para lanzar la ejecución del `jar` (recomendado antes de enviarlo a `AULES`) debes seguir estas instrucciones:

Debes ejecutar el siguiente comando en la consola, sustituyendo:

- #RUTA_JAVA_FX: por la ruta absoluta donde se encuentran los archivos jar de javaFx
- #NOMBRE_DE_TU_JAR: nombre del jar que has generado en el paso anterior

```
java --module-path #RUTA_JAVA_FX --add-modules javafx.controls,javafx.fxml -jar #NOMBRE_DE_TU_JAR.jar
```

Envia el fichero `JAR` y un `pdf` explicando las partes que te han parecido más complicadas al realizar la práctica.

8. Píldoras informáticas relacionadas

- https://www.youtube.com/playlist?list=PLNjWMbvTJAIjLRW2qyuc4DEgFVW5YFRSR
- https://www.youtube.com/playlist?list=PLaxZkGlLWHGUWZxuadN3J7KKaICRlhz5-

9. Fuentes de información

- Apuntes de Jose Antonio Diaz-Alejo
- https://github.com/openjfx/openjfx-docsopen
- https://github.com/openjfx/samples
- FXDocs
- https://openjfx.io/openjfx-docs/
- https://docs.oracle.com/javase/8/javafx/user-interface-tutorial
- https://github.com/JonathanGiles/scenic-view