Taller UD09_03: 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`.

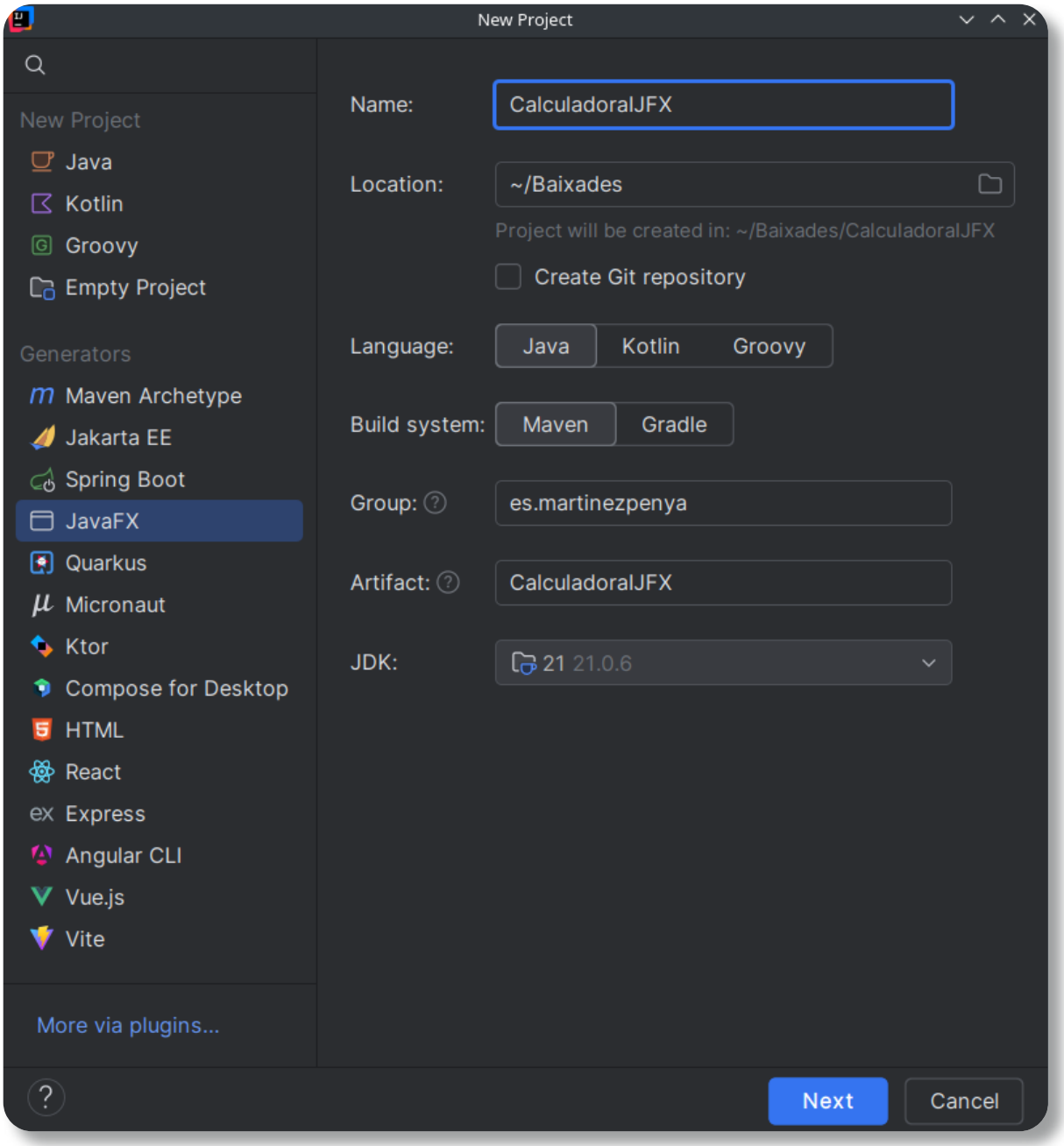
Necesitaras:

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

2. Crear proyecto

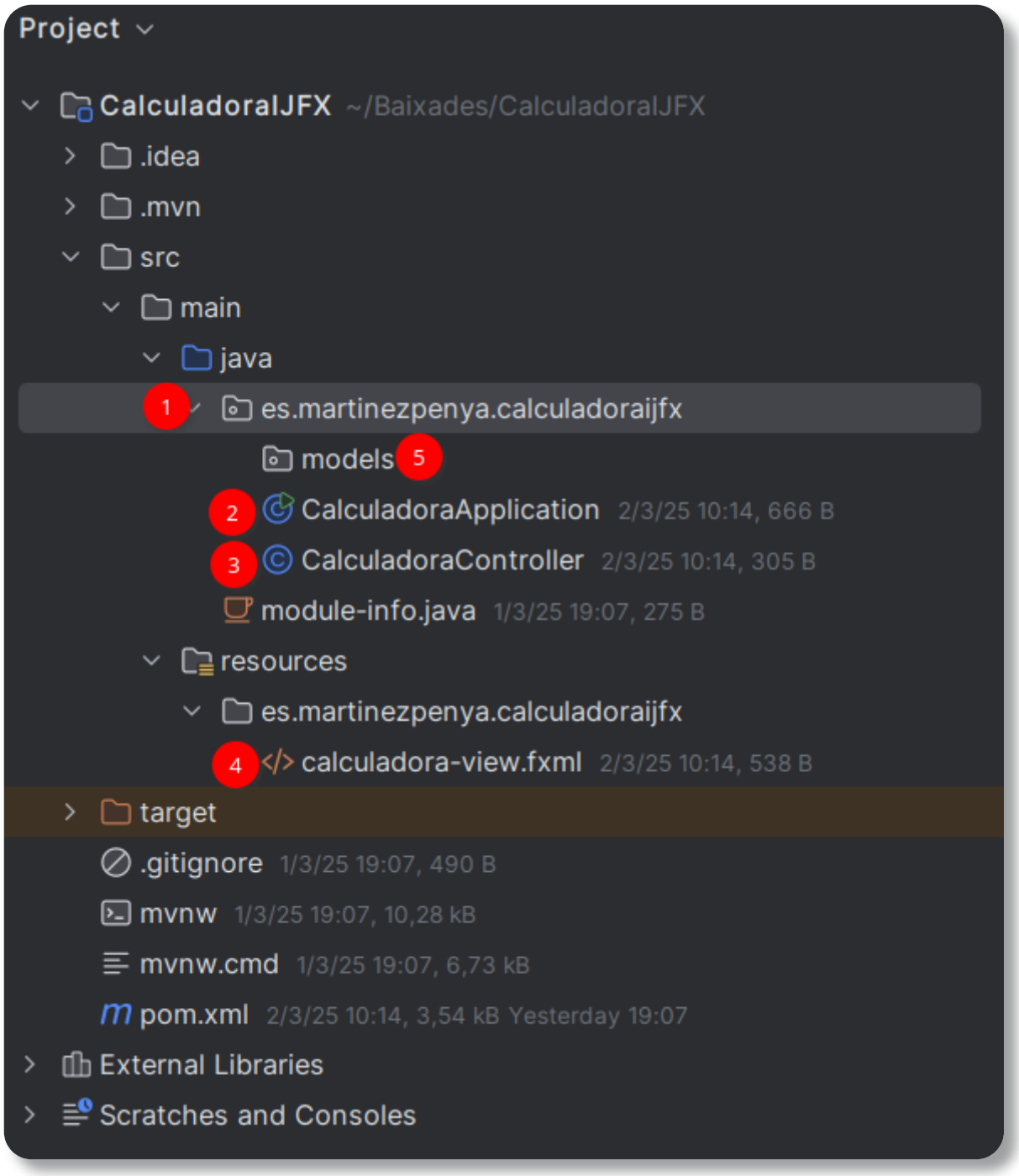
Vamos a crear el proyecto con el template de `JavaFX`.

En nuestro caso llamamos al proyecto `CalculadoraIJFX`, yo como Group pongo "`es.martinezpenya`", pero tu puedes poner tu "dominio" (tus apellidos) y recuerda en la siguiente pantalla marcar: `ControlsFX` y `FormsFX`



El asistente de IntelliJ sigue el modelo **MVC**, pero vamos a renombrar las clases que crea por defecto y crearemos un package (carpeta) donde incluiremos los modelos (el proyecto por defecto es tan simple que no tiene).

Cuando hagas todo eso tu proyecto debería tener un aspecto similar a este::



Si lanzamos la aplicacion `CalculadoraApplication` (que es la única clase con `main`) solo deberíamos ver una ventana con un botón **Hello**, y si lo pulsamos aparece el texto **"Welcome to JavaFX Application!"**

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 `models` con el siguiente contenido:

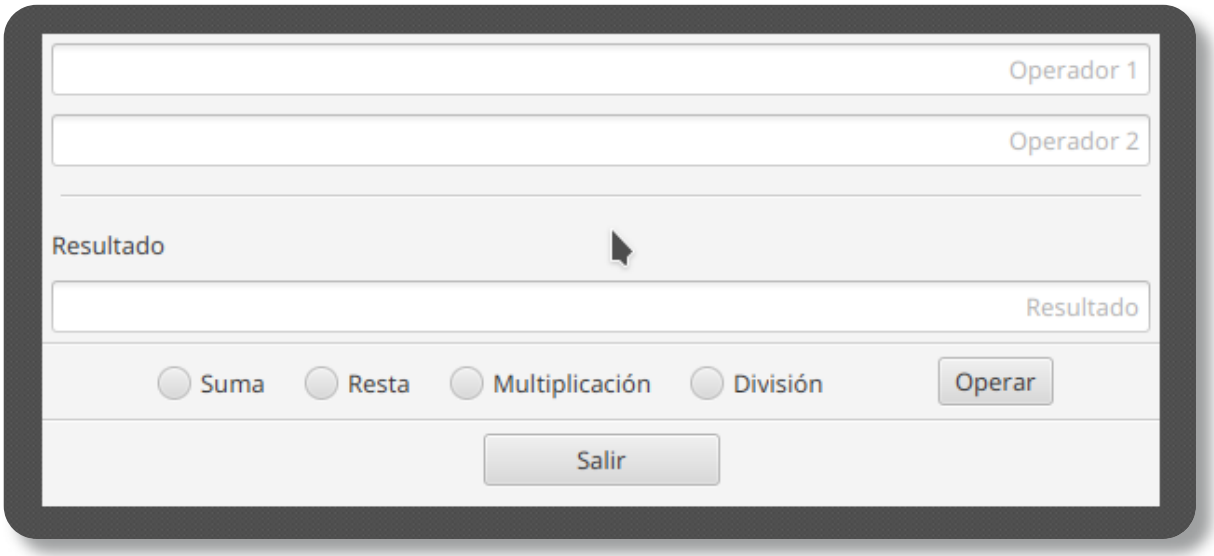
```java
package es.martinezpenya.calculadoraijfx.models;

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 (`suma`, `resta`, `multiplicacion` y `division`).

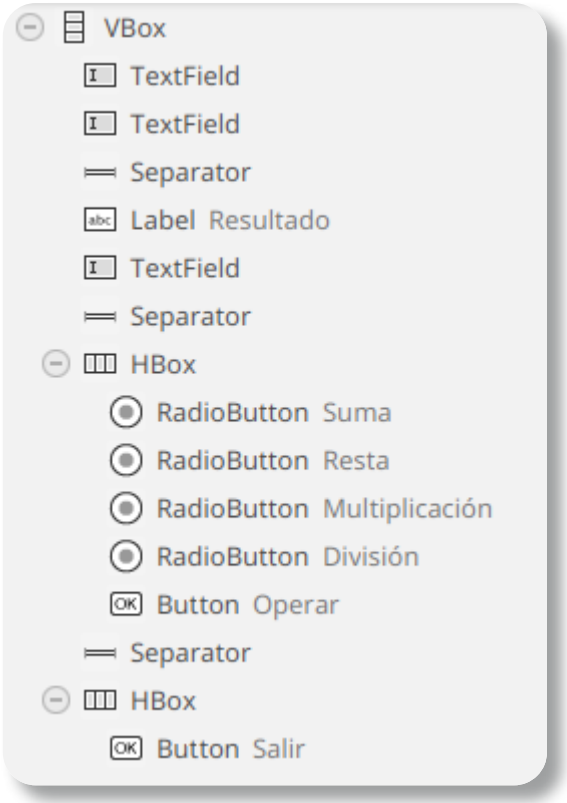
4. Vista

Ahora vamos a editar el fichero `calculadora-view.fxml` del paquete `resources`. Para ello, pulsamos el botón derecho sobre el nuevo fichero y elegimos la opción `Open in SceneBuilder` por defecto y el botón **Hello**.

Ahora, usando contenedores y componentes básicos deberías crear una ventana similar a esta:



Este ejemplo tiene la siguiente jerarquía:



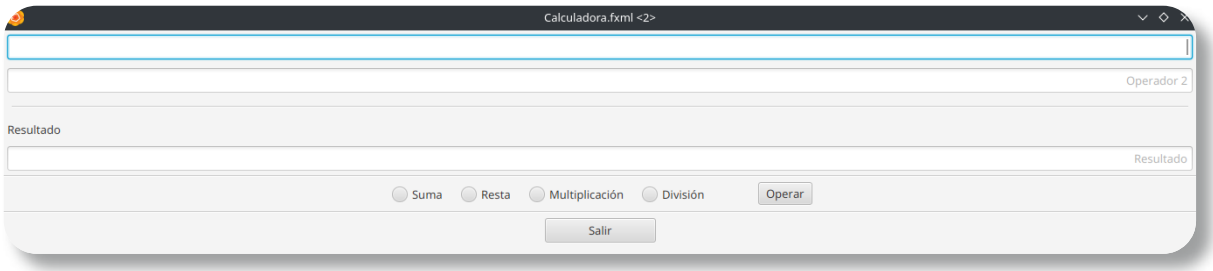
Recuerda dar nombre a todos los componentes en la pestaña `code` al campo `fx:id`.

`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`

Importante Puede que tengas errores en el fichero `fxml`, de momento es normal, continua con el proceso.

5. Controladores

5.1. `CalculadoraController.java`

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

Añade el texto `fx:controller="es.martinezpenya.calculadoraijfx.CalculadoraController"` al final de la linea del `VBOX` del archivo `calculadora-view.fxml`:

```
<VBox maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="253.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/21"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="es.martinezpenya.calculadoraijfx.CalculadoraController">
```

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

> **Importante** Es posible que al generar el código para el controlador, no se agreguen automáticamente los imports, debes hacerlo tu a partir de la ayuda que te proporciona el IDE, pero recuerda que todos los imports deben estar relacionados con `JavaFx` y no con `AWT` (Antigüa tecnología de Java para interfaces gráficas)

Ahora, dentro del `CalculadoraController.java` debemos primero eliminar la acción que traía el código de ejemplo:

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

Y agregaremos el código necesario para gestionar las acciones de los botones, y además asegurarnos que los *radio buttons* son auto-excluyentes:

Acción `salir`:

```
    @FXML
    public void salir(ActionEvent actionEvent) {
        Stage stage = (Stage) btnSalir.getScene().getWindow();
        stage.close();
    }
```

Acción `operar`:

```
    @FXML
    public void operar(ActionEvent actionEvent) {
        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 es.martinezpenya.calculadoraijfx.models.Operaciones;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
```

Acción `initialize`:

```
@FXML
public void initialize() {
    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 modificar la clase `CalculadoraApplication`, que contendrá el método `main` que lanzará la aplicación JavaFX. Una vez modificado (sobre todo el título y tamaño de la ventana), debería quedar algo parecido a esto:

```
package es.martinezpenya.calculadoraijfx;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class CalculadoraApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(CalculadoraApplication.class.getResource("calculadora-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 600, 253);
        stage.setTitle("Calculadora de David");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

Fíjate como se carga el recurso desde una "ruta" relativa a partir del nombre de la clase. Te dejo aquí debajo un cuadro resumen con las diferentes maneras de cargar recursos en Java.

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

A comparison of different ways of resources loading in Java

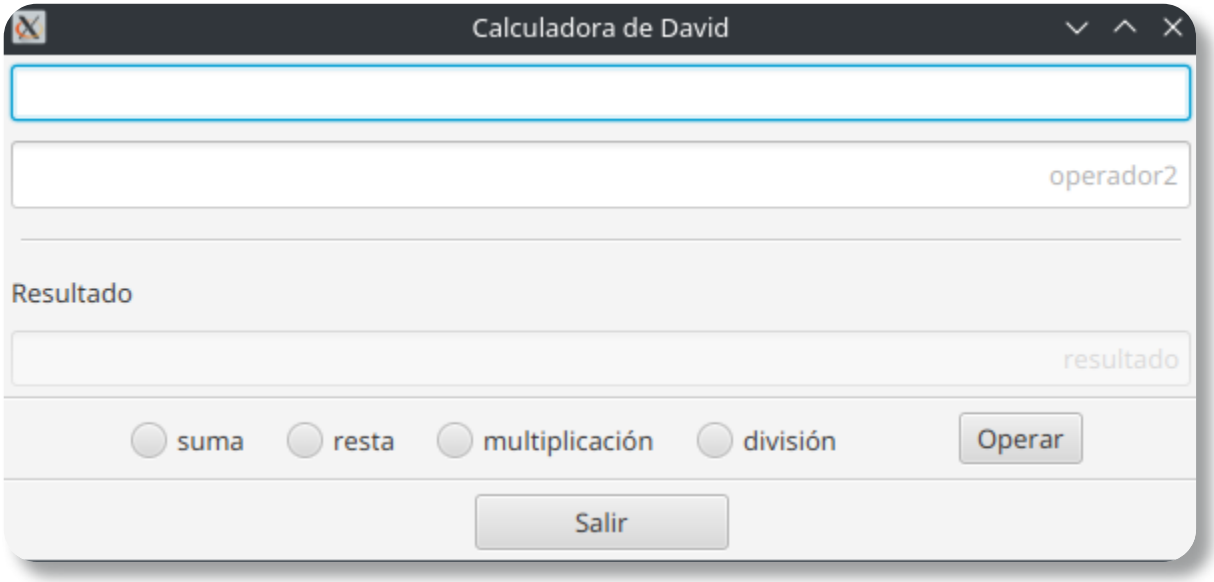
Followings are the preferred ways to load resources in classpath.

- `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

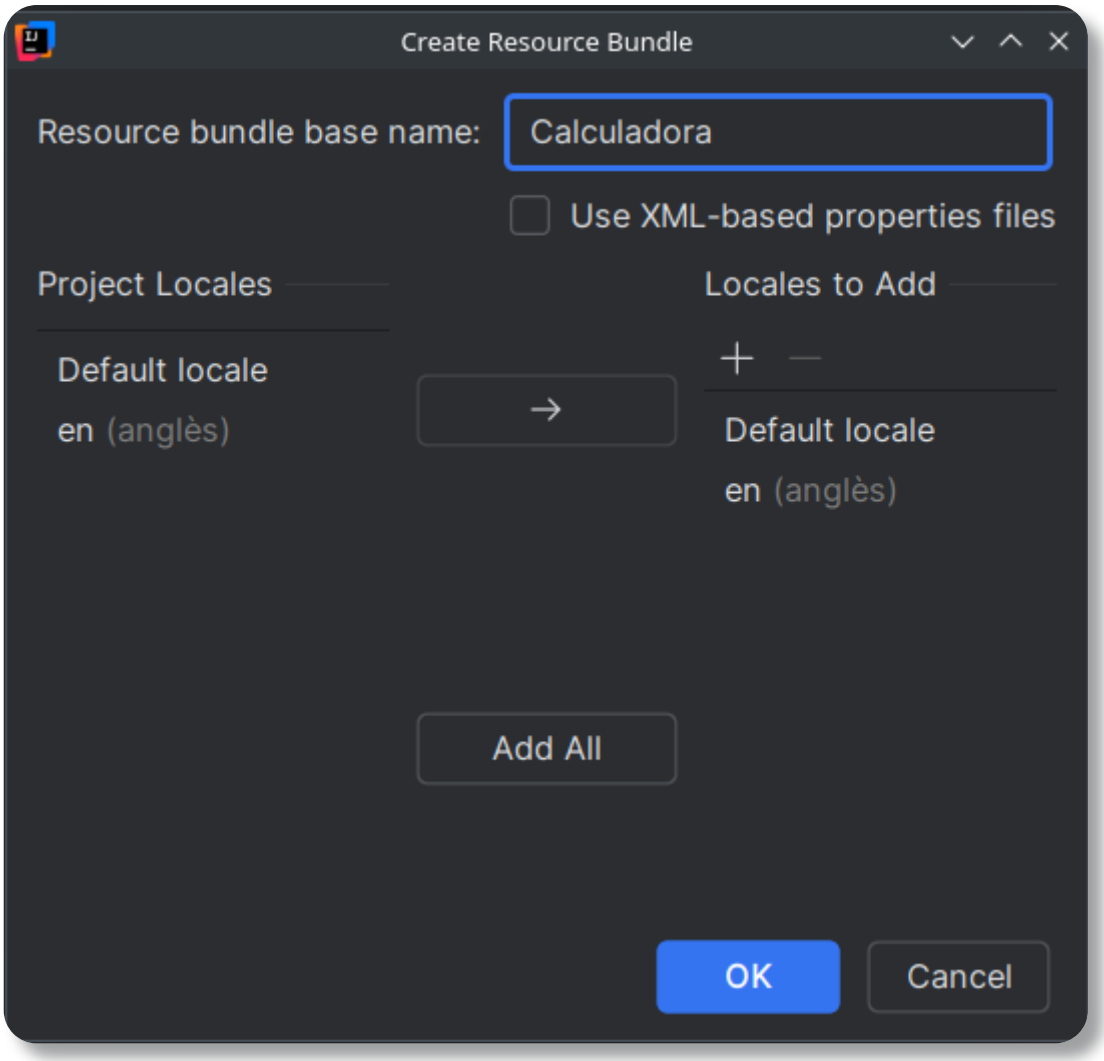
Si todo ha ido bien debería aparecer nuestra calculadora en pantalla:



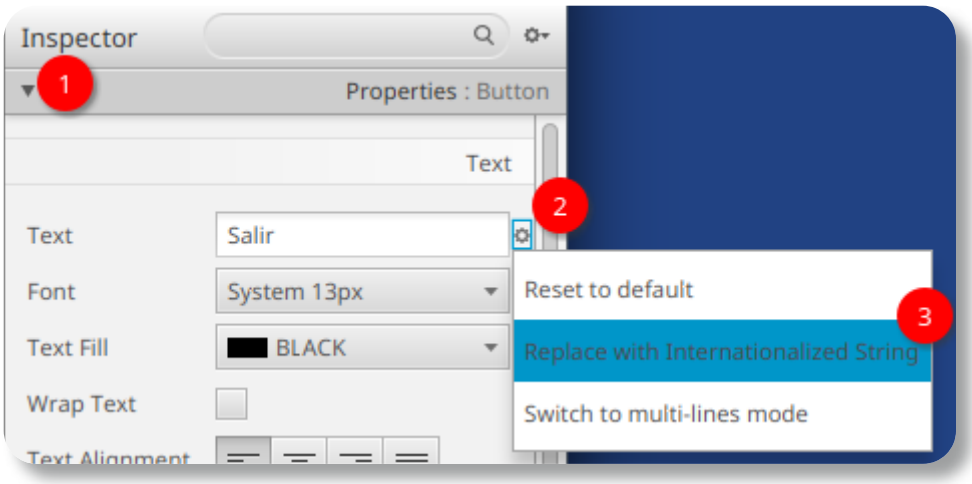
7. Internacionalización `I18N` del formulario (ampliación voluntaria)

En las aplicaciones de interfaz gráfica es muy recomendable (y relativamente fácil) añadir la funcionalidad multi-idioma. Para ilustrar como se hace ayudados de `SceneBuilder` y `JavaFX`, haremos que nuestra calculadora funcione en Español e Inglés.

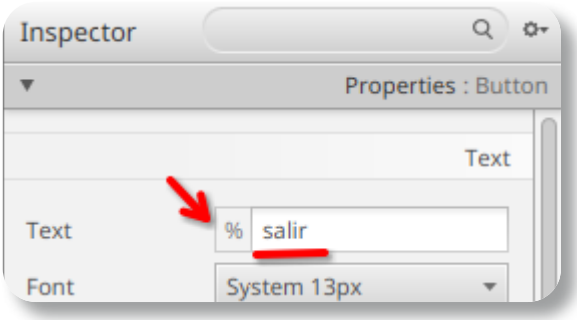
Dentro del paquete de recursos (resources) vamos a añadir un `Resource Bundle`, este tipo de recurso siempre acaba con la cadena "`.properties`" para que IntelliJ lo reconozca como un recurso de propiedades, así que nosotros pondremos solo `Calculadora`, y el IDE añadirá el "`.properties`". Consideraremos que el lenguaje por defecto será el Español, así que solo debemos añadir un lenguaje más que será el "Ingles" (en). Por último podemos elegir guardar las propiedades en un fichero xml o un fichero de texto plano (desmarcando la opción):



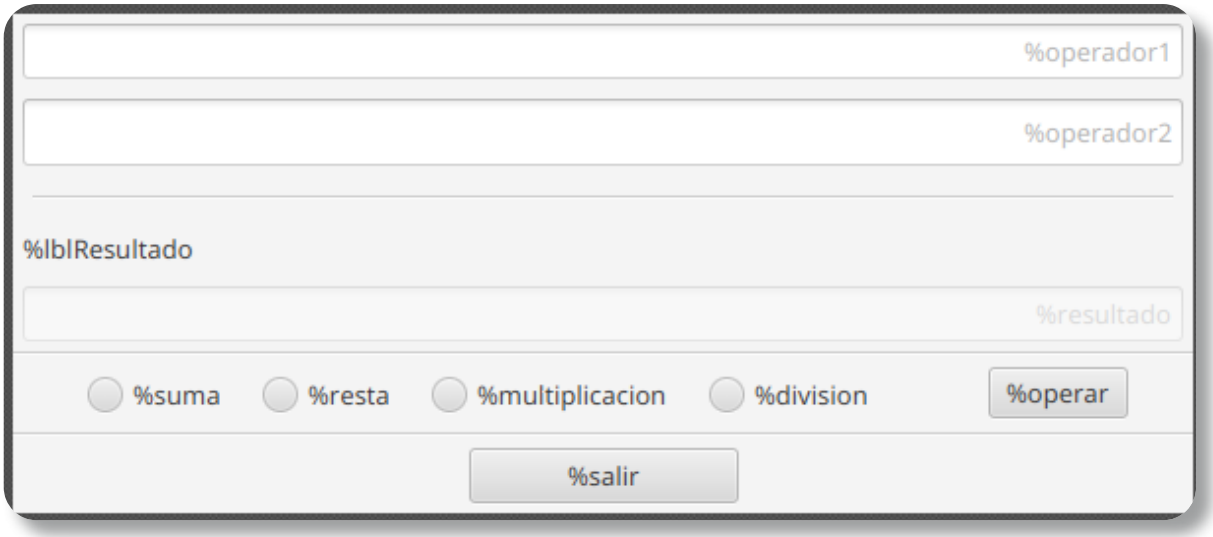
Lo siguiente será convertir todas las cadenas de texto de la interfaz gráfica en "Cadenas internacionalizadas". Abre el formulario `fxml` con `SceneBuilder`, elige por ejemplo el botón `Salir`, y en la propiedad `Text` (dentro de `Properties`) elige la rueda dentada que aparece a la derecha y del menú que aparece elige la opción "Replace with Internationalized String":



Ahora el campo `Text` deberías dejarlo así:

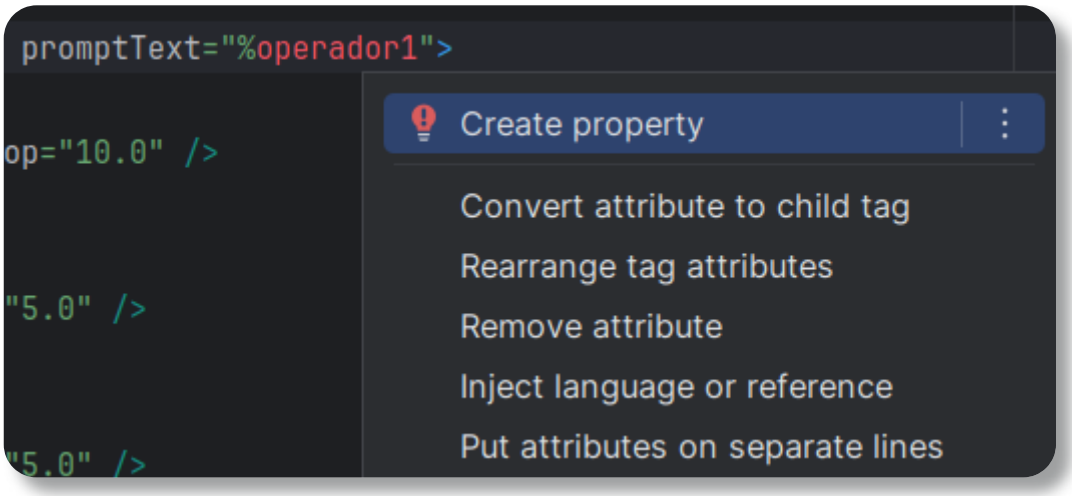


Repite este proceso con todas las partes de la interfaz gráfica que deban estar en dos idiomas.

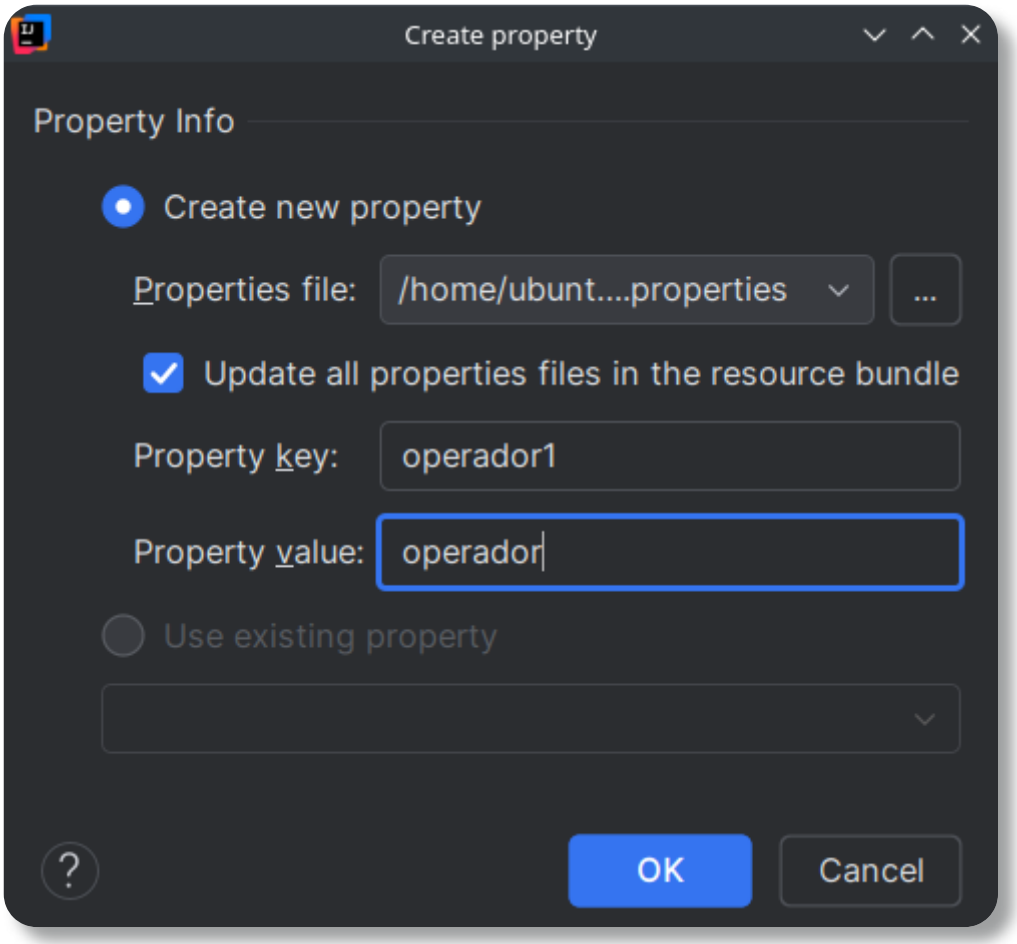


Ahora veras que el código fuente del fichero `fxml` contiene errores en cada una de las lineas que hacen referencia a estas nuevas cadenas de texto internacionales que hemos definido, eso de momento está bien.

Ahora, usando el asistente de IntelliJ, le pedimos que nos ayude a corregir los errores:

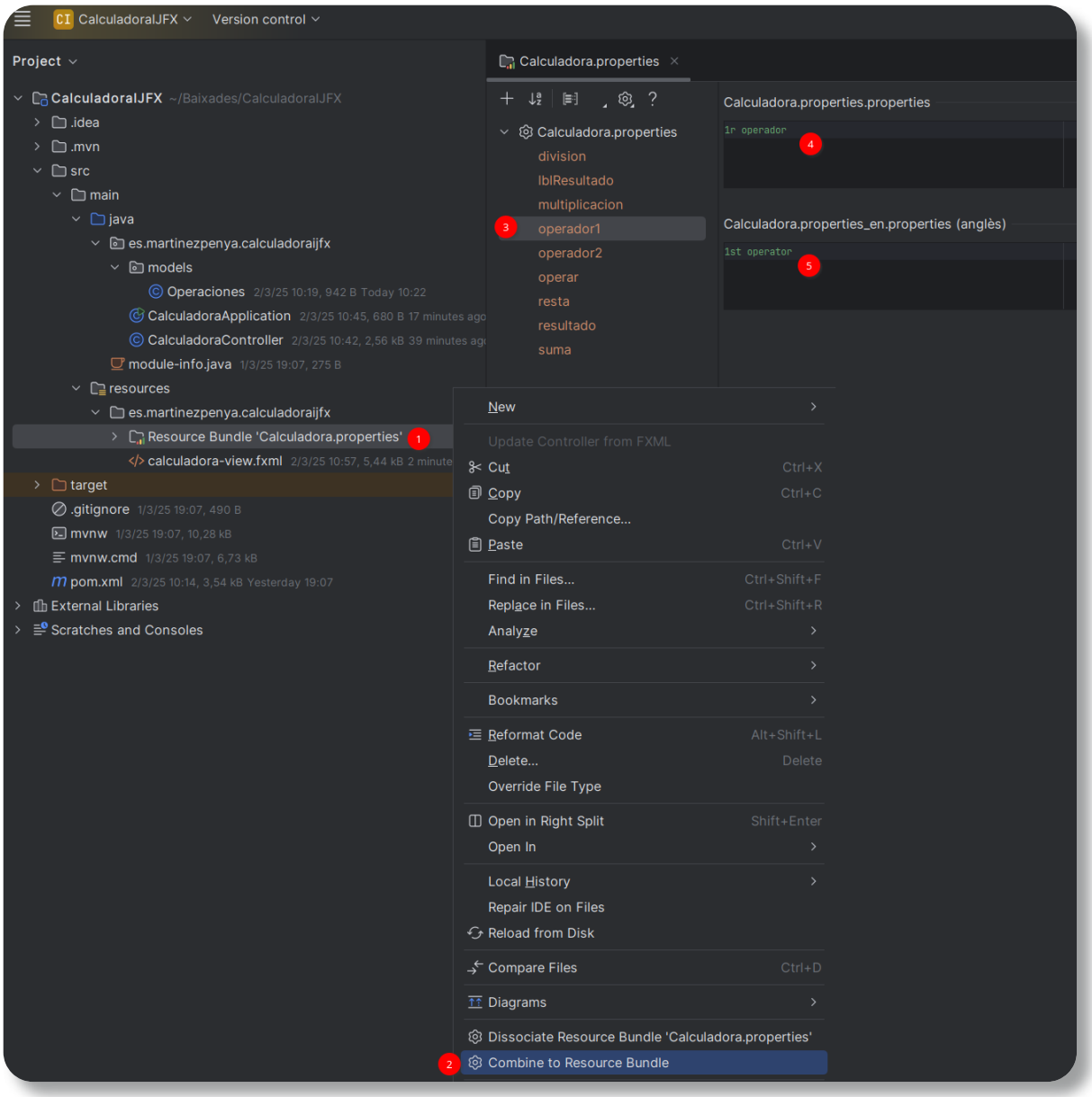


Y a continuación (si no te aparece directamente el fichero de propiedades es que no has hecho bien el primer paso):



Repite el paso anterior para todas las propiedades del FXML que dan error, genera todas las etiquetas con el idioma por defecto (Español).

Ahora solo queda añadir las diferentes traducciones al Inglés, para ello combinaremos todas las propiedades en un único editor:



Para ello:

1. Pulsa botón derecho sobre el recurso
2. Elige la opción "Combine to Resource Bundle"
3. Elegimos la "etiqueta" que queremos traducir
4. Escribimos/modificamos el idioma por defecto (Español)
5. Escribimos/modificamos el idioma en Ingles.

Ahora debemos hacer que el main de la clase `CalculadoraApplication.java` cargue estos recursos al cargar el formulario. Añadiremos la linea del recurso (ResourceBundle) y añadiremos el mismo (bundle) al cargar el formulario:

```
1  public void start(Stage stage) throws IOException {
2      ResourceBundle bundle = ResourceBundle.getBundle(getClass().getPackage().getName()+"."
   + "Calculadora");
3      FXMLLoader fxmlLoader = new
   FXMLLoader(CalculadoraApplication.class.getResource("calculadora-view.fxml"), bundle);
4      Scene scene = new Scene(fxmlLoader.load(), 600, 253);
5      stage.setTitle("Calculadora de David");
6      stage.setScene(scene);
7      stage.show();
8  }
```

Ahora, al lanzar la aplicación, aparecerá en el idioma español por defecto, a no ser que nuestro sistema esté configurado en Inglés. Podemos forzar que se ejecute por ejemplo siempre en inglés añadiendo las siguientes `VM Options` al entorno de ejecución del proyecto:

```
-Duser.country=UK -Duser.language=en
```

8. Internacionalización de los mensajes de error (ampliación voluntaria)

En el punto anterior hemos traducido el formulario a dos idiomas, pero los mensajes de error siguen saliendo solo en Español. Para corregirlo necesitamos añadir más propiedades al `ResourceBundle` "**Calculadora**" (al menos una para el título de las ventanas de Error, y otras dos con los mensajes que necesita el controlador de la Calculadora, "formato incorrecto de los operadores" y "el operador 2 no puede ser 0"). Una vez creadas las propiedades, solo debemos añadir al controlador la linea para que cargue los recursos:

```
ResourceBundle bundle = ResourceBundle.getBundle(getClass().getPackage().getName()+"." +
"Calculadora");
```

Y luego cambiar el método `operar` para que haga uso de ellas en lugar de los Strings:

```
@FXML
public void operar(ActionEvent actionEvent) {
    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(bundle.getString("error"));
                alert.setContentText(bundle.getString("operador2Non0"));
                alert.showAndWait();
            }
        }
    } catch (NumberFormatException numberFormatException) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(null);
        alert.setTitle(bundle.getString("error"));
        alert.setContentText(bundle.getString("formatError"));
        alert.showAndWait();
    }
}
```

En mi ejemplo las propiedades del `ResourceBundle` se llaman `error`, `operador2Non0` y `formatError`.

9. Tarea Aules

La tarea consiste en seguir la guía y crear tu aplicación en IntelliJ, pon como título de aplicación tu nombre y apellidos.

Envía el proyecto en `zip` y un `pdf` explicando las partes que te han parecido más complicadas al realizar la práctica, que parte te ha parecido más interesante, cual te gustaría ampliar y alguna captura mostrándola en funcionamiento.

10. Píldoras informáticas relacionadas

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

11. 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
- https://blog.devgenius.io/flutter-internationalization-i18n-with-getx-d3a6465d1282