

# Programación de Interfaces Gráficas en Java

## Objetivo: Programar aplicaciones usando

## IntelliJ y JavaFX

ELO329: Diseño y Programación Orientados a Objetos

Departamento de Electrónica

Universidad Técnica Federico Santa María

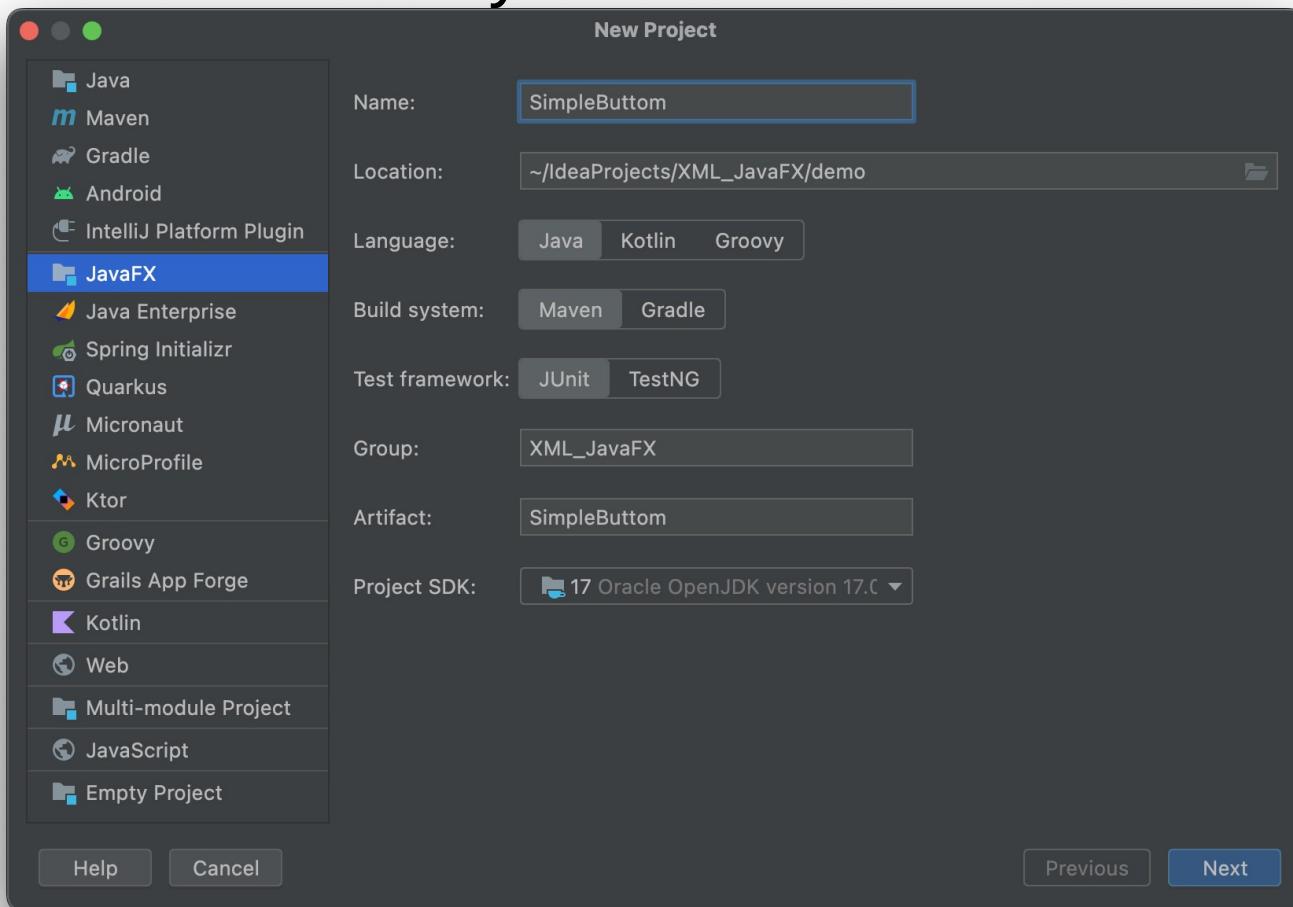
Este material ha sido preparado desde varias fuentes, entre ellas el material  
del Dr. Paul Fodor, Stony Brook University

# Aplicaciones Gráfica en IntelliJ

- ❑ Podemos mencionar dos formas para desarrollar Interfaces Gráficas de Usuario en IntelliJ:
  - Especificación de la interfaz gráfica en XML (Extensible Markup Language o Lenguaje de Marcas Extensible)
  - Uso de paquetes Java solamente
- ❑ El uso de XML tiene al menos dos virtudes:
  - Separa el diseño de la interfaz de su programación
  - Existe software para que diseñadores puedan crear la apariencia visual de una aplicación (Ej. [Scene Builder](#))
- ❑ El uso de solo Java tiene como ventajas:
  - La lógica de la interfaz sigue los fundamentos de Java
  - Da mayor flexibilidad para desarrollar interfaces gráficas (Ej. caso de tarea 2)

# Desarrollo usando XML y JavaFX en IntelliJ

- ❑ Paso 1: Abra IntelliJ IDEA y cree un nuevo proyecto JavaFX agregando un nombre y ubicación



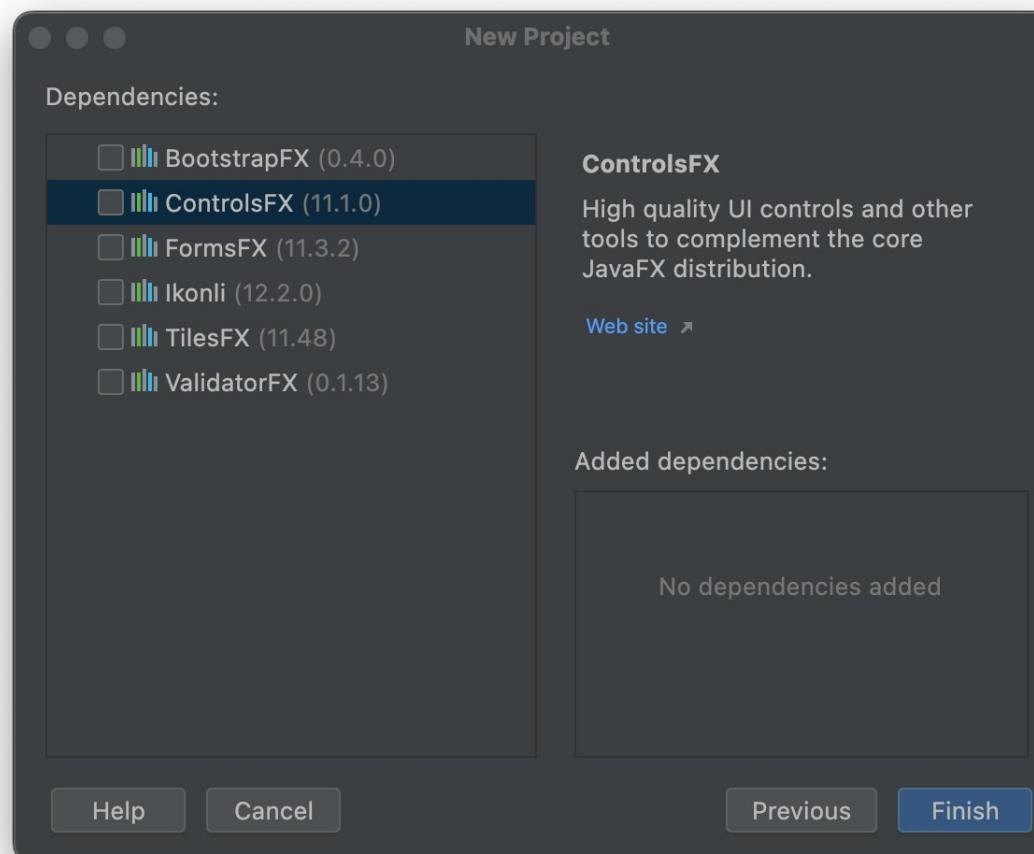
**Maven:** es la herramienta para la gestión de proyectos

**Junit:** Conjunto de bibliotecas para hacer pruebas unitarias

**Group:** es el paquete

# Desarrollo usando XML y JavaFX en IntelliJ

- ❑ Paso 2: Para aplicaciones simples no requiere dependencias y ya puede compilar su primer programa.



# Desarrollo usando XML y JavaFX en IntelliJ

- Se aprecia la carga de la interfaz gráfica con FXMLLoader.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure for "SimpleButton" located at "/IdeaProjects/XML\_J". It includes a .idea folder, a src directory with main/java/xml javafx/simplebutton (containing HelloApplication.java and HelloController.java) and module-info.java, a resources directory with pom.xml and SimpleButton.iml, and External Libraries and Scratches and Consoles.
- Code Editor:** The file "HelloApplication.java" is open, displaying Java code to load an FXML view and show a stage. The code uses FXMLLoader to load "hello-view.fxml" and sets the stage title to "Hello!".
- Run Preview:** A preview window titled "Hello!" shows a JavaFX application window with the title "Hello!". Inside the window, the text "Welcome to JavaFX Application!" is displayed, and there is a button labeled "Hello!".
- Bottom Status Bar:** Shows "Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK and Maven library shared indexes // Always download // Download once // ... (a minute ago)" and "Event Log" with 1 entry.

- Veamos qué contiene el archivo hello-view.fxml ...

# Desarrollo usando XML y JavaFX en IntelliJ

## □ Descripción de la interfaz en XML

The screenshot shows the IntelliJ IDEA interface with the project 'SimpleButton' open. The 'hello-view.fxml' file is selected in the Project tool window. The code editor displays the FXML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
       fx:controller="xml_javafx.simplebutton.HelloController">
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>
    <Label fx:id="welcomeText"/>
    <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```

Annotations explain specific parts of the code:

- Administrador del espacio: Points to the import statements for Insets, Label, and VBox.
- Atributos VBox: Points to the 'alignment', 'spacing', and 'fx:controller' attributes of the VBox root node.
- Nodos dentro del Vbox con el nombre para accederlo: Points to the 'padding' and 'Label' nodes within the VBox.
- Con atributo action se especifica nombre del manejador "handler": Points to the 'onAction="#onHelloButtonClick"' attribute of the Button node.

At the bottom of the code editor, it says "Editor gráfico de la interfaz".

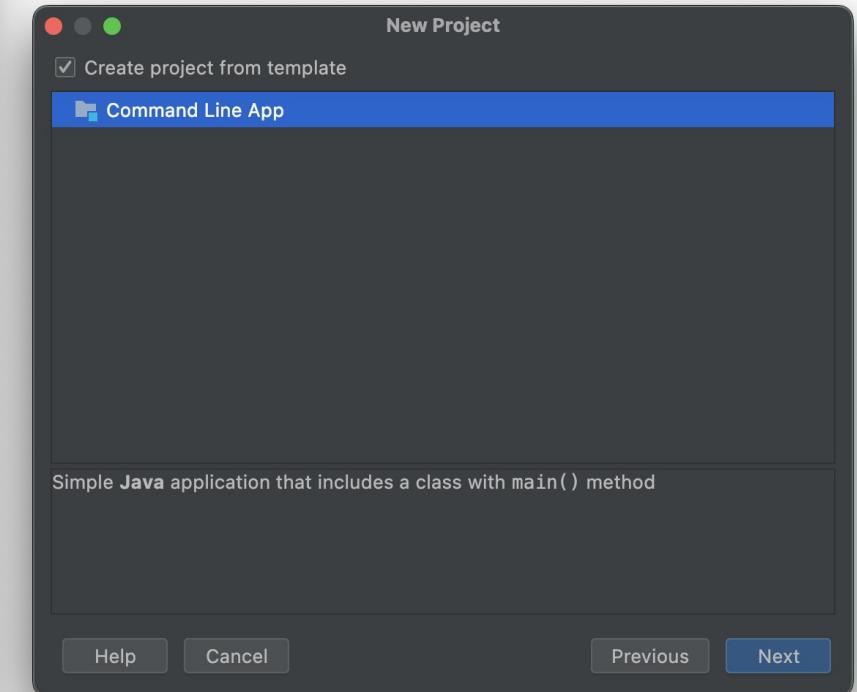
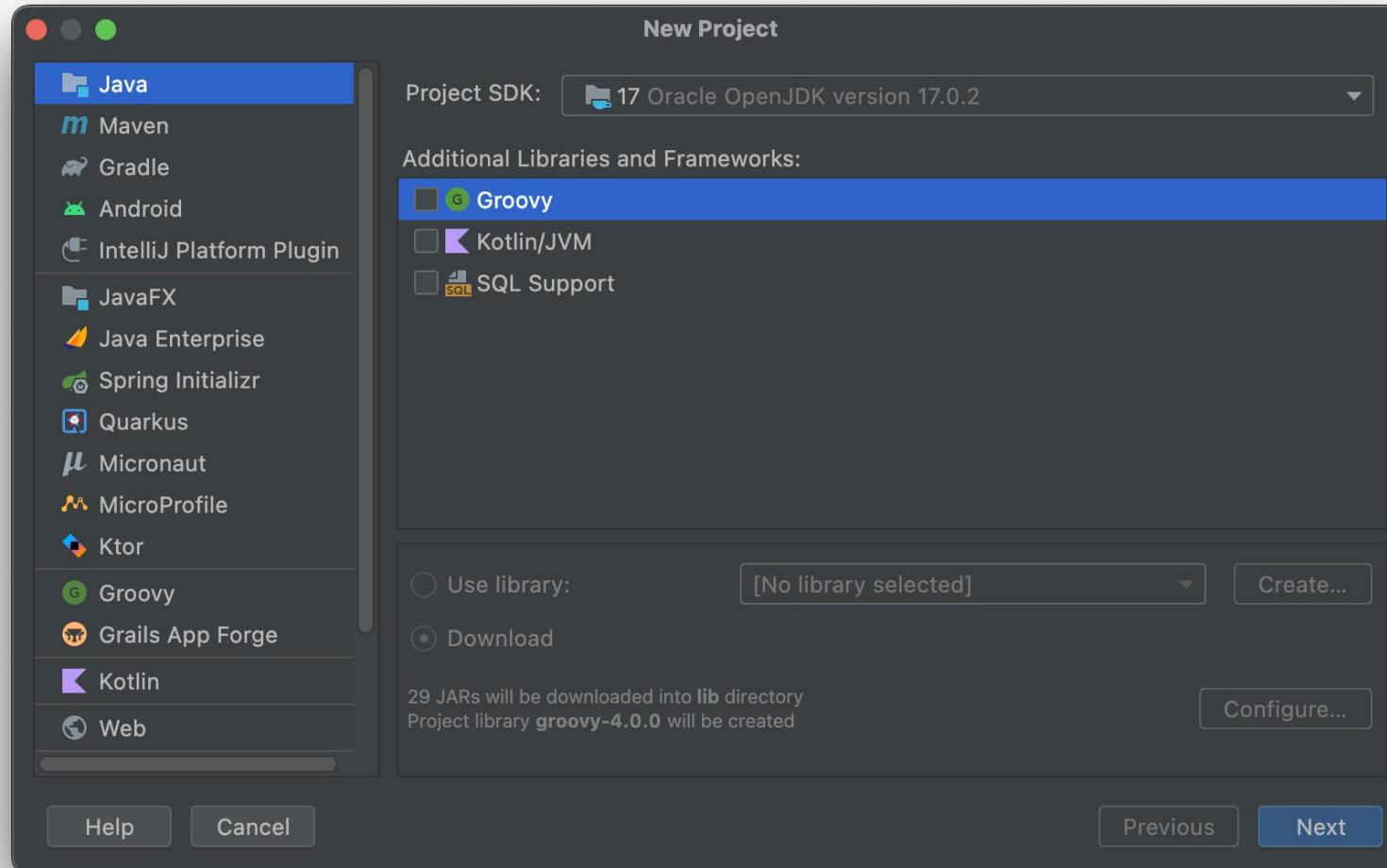


# Scene Builder

- ❑ JavaFX provee un entorno para creación de interfaces de usuario: el Scene Builder.
- ❑ Este se puede descargar desde la página  
<https://gluonhq.com/products/scene-builder/>
- ❑ En este curso (tarea 2) usted desarrollará su propia interfaz gráfica **codificando en Java con JavaFX (sin XML)** para que usted vea la relación entre clases y el resultado gráfico.
- ❑ Superada la etapa previa, usted puede usar Scene Builder pero no será exigido en este curso.

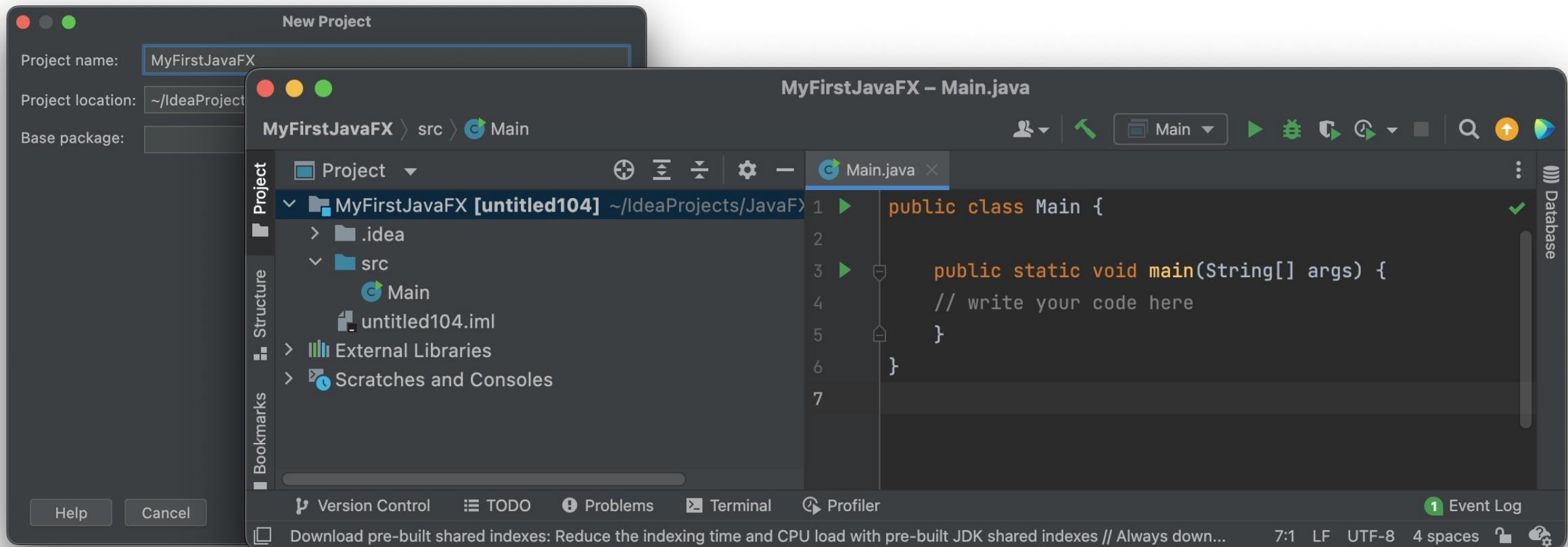
# Aplicación solo Java en IntelliJ

- ❑ Paso 1: Seleccione Java en su proyecto y luego tipo línea de comando



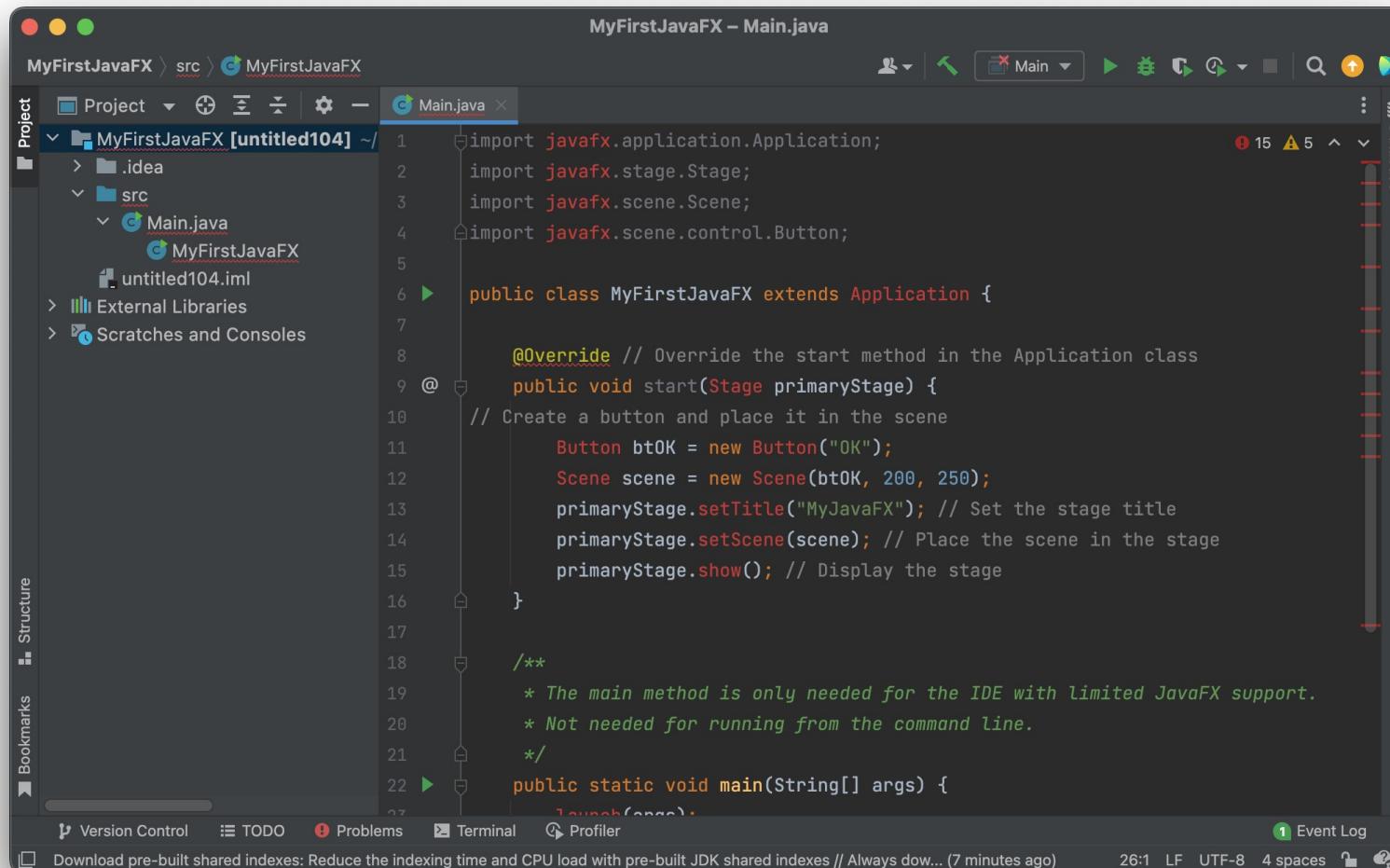
# Aplicación solo Java en IntelliJ

## ❑ Paso 2: Asigne un nombre a su proyecto



# Aplicación solo Java en IntelliJ

- ❑ Paso 3: Cree el código de su aplicación (por ejemplo copy-paste de gitlab).



The screenshot shows the IntelliJ IDEA interface with a JavaFX application named "MyFirstJavaFX". The project structure on the left includes ".idea", "src" (containing "Main.java" and "MyFirstJavaFX"), "untitled104.iml", "External Libraries", and "Scratches and Consoles". The main editor window displays the "Main.java" code:

```
MyFirstJavaFX – Main.java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;

public class MyFirstJavaFX extends Application {

    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a button and place it in the scene
        Button btOK = new Button("OK");
        Scene scene = new Scene(btOK, 200, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

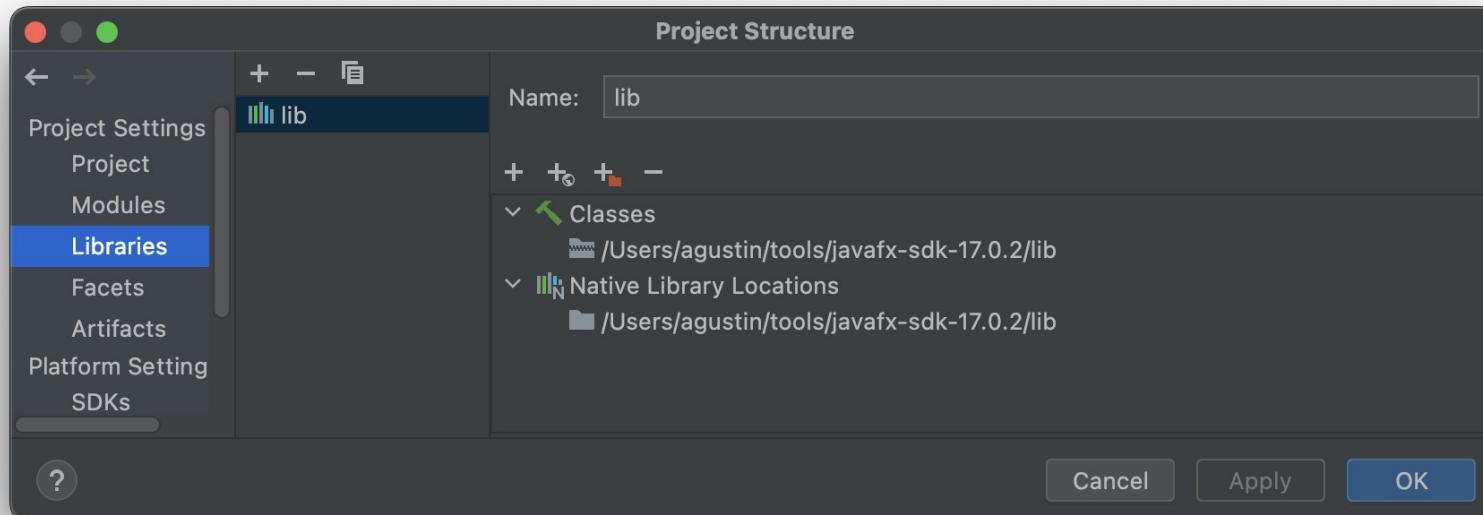
    /**
     * The main method is only needed for the IDE with limited JavaFX support.
     * Not needed for running from the command line.
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

The code uses JavaFX classes and annotations, with some parts highlighted in red (e.g., "javafx" package, "Application", "Stage", "Scene", "Button", "@Override", "start", "Scene", "setTitle", "setScene", "show", "main"). The status bar at the bottom indicates "Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always dow... (7 minutes ago)".

- IntelliJ no reconoce las clases... (en rojo)
- Debemos indicar dónde están (biblioteca JavaFX)

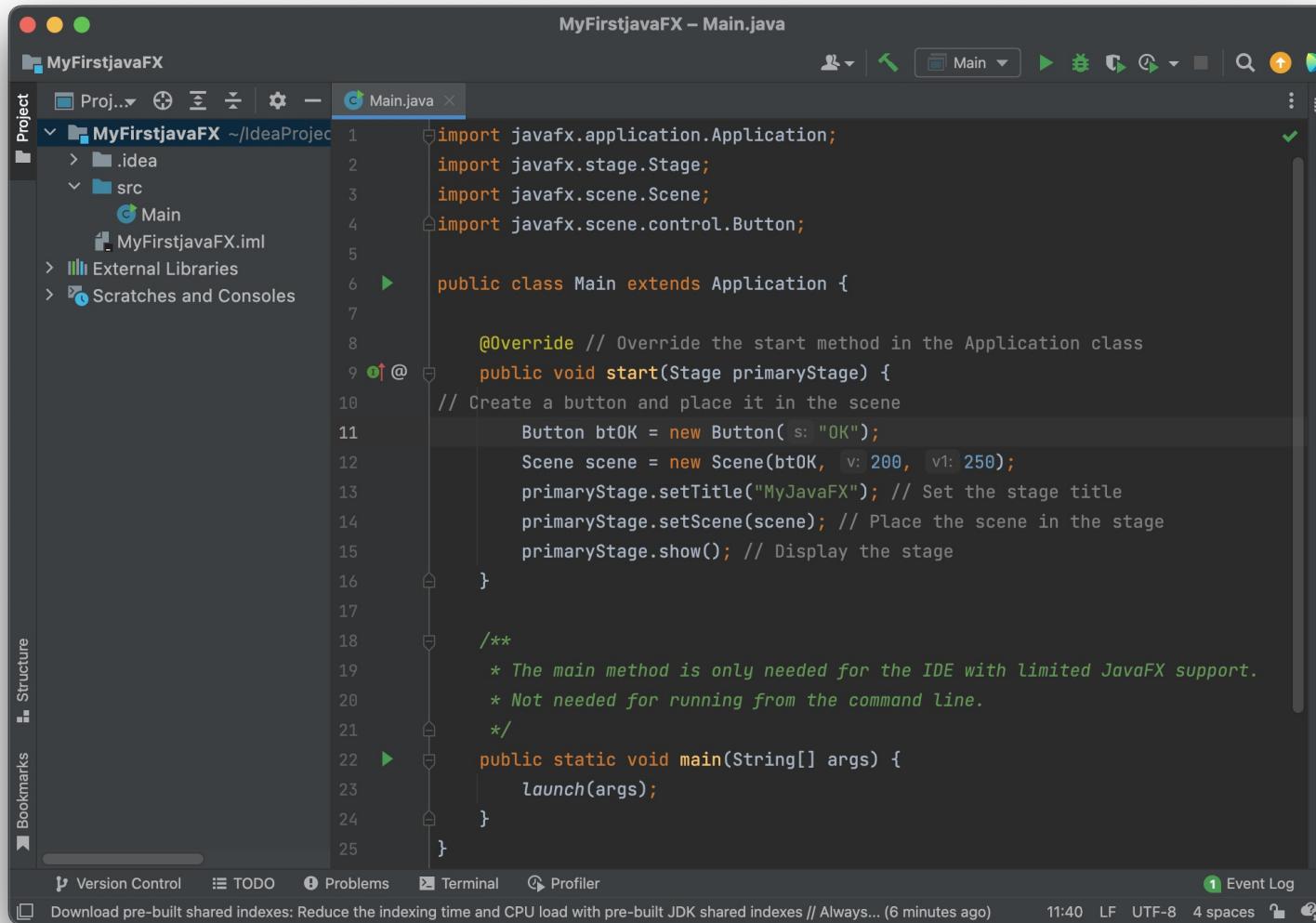
# Aplicación solo Java en IntelliJ

- ❑ Paso 4: Configuración de biblioteca JavaFX (en analogía a –module-path ...)
  - Botón derecho sobre nombre del proyecto MyFirstJavaFX -> Open Module Settings
  - Seleccione Libraries a la izquierda, luego +, y seleccione directorio lib donde usted descargó JavaFX, Seleccione OK.



# Aplicación solo Java en IntelliJ

- ❑ Si lo previo estuvo bien, ahora su código se ve así (notar que las clases no están en rojo):



The screenshot shows the IntelliJ IDEA interface with the following details:

- Title Bar:** MyFirstjavaFX – Main.java
- Toolbars:** Standard IntelliJ toolbar with icons for file operations, search, and navigation.
- Project View:** Shows the project structure under "MyFirstjavaFX". It includes a ".idea" folder, a "src" folder containing a "Main" class, and files "MyFirstjavaFX.iml", "External Libraries", and "Scratches and Consoles".
- Code Editor:** Displays the Main.java code. The code initializes a JavaFX Application, creates a button, and sets up a stage with a title and scene. It also includes a main method for command-line execution.
- Status Bar:** Shows system information like "Event Log", "Version Control", "TODO", "Problems", "Terminal", "Profiler", and "Download pre-built shared indexes".
- Bottom Status:** Shows the date and time as "11:40 LF UTF-8 4 spaces" and a download progress bar for "Download pre-built shared indexes".

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;

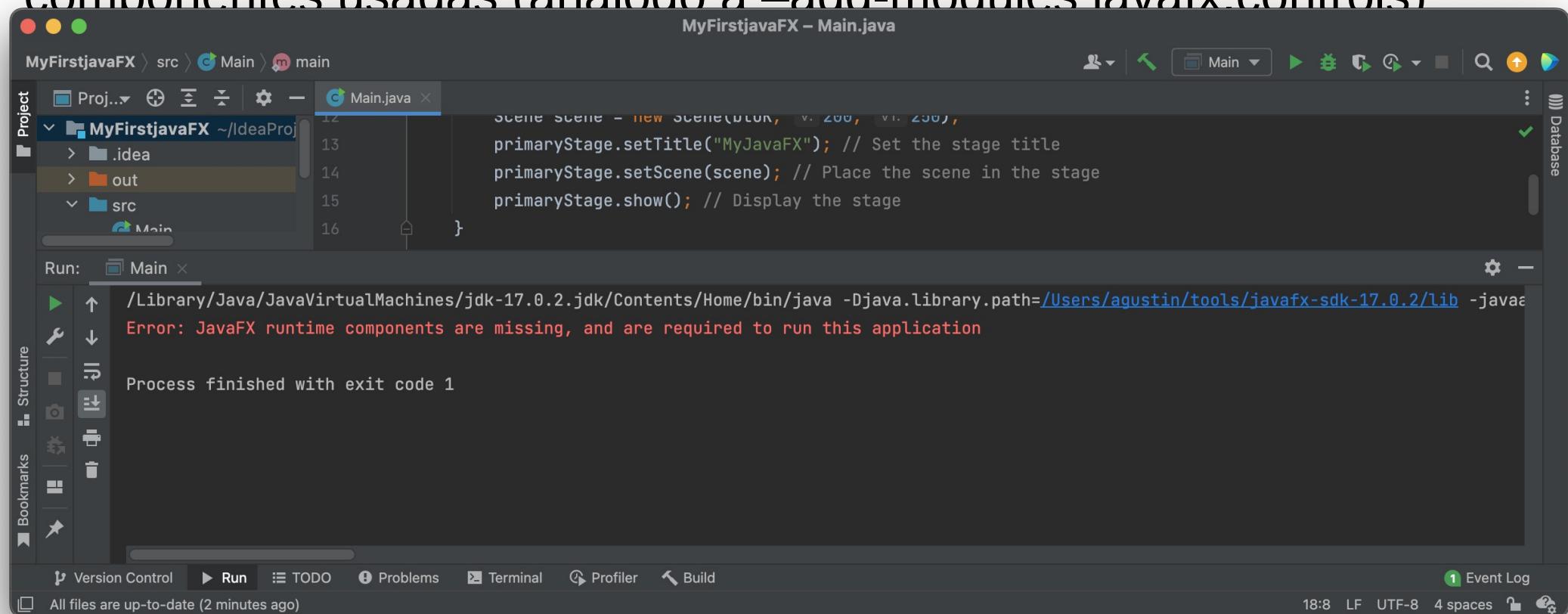
public class Main extends Application {

    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a button and place it in the scene
        Button btOK = new Button("OK");
        Scene scene = new Scene(btOK, 200, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

    /**
     * The main method is only needed for the IDE with limited JavaFX support.
     * Not needed for running from the command line.
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

# Compilación y ejecución

- Compilación: presione el martillo en lado superior derecho, debería ser OK
- Ejecución: debería arrojar un error: no hemos especificado las componentes usadas (análogo a `--add-modules javafx.controls`)



The screenshot shows the IntelliJ IDEA interface with a JavaFX project named "MyFirstjavaFX". The "Main.java" file is open in the editor, containing the following code:

```
Scene scene = new Scene(null, 200, 200);
primaryStage.setTitle("MyJavaFX"); // Set the stage title
primaryStage.setScene(scene); // Place the scene in the stage
primaryStage.show(); // Display the stage
```

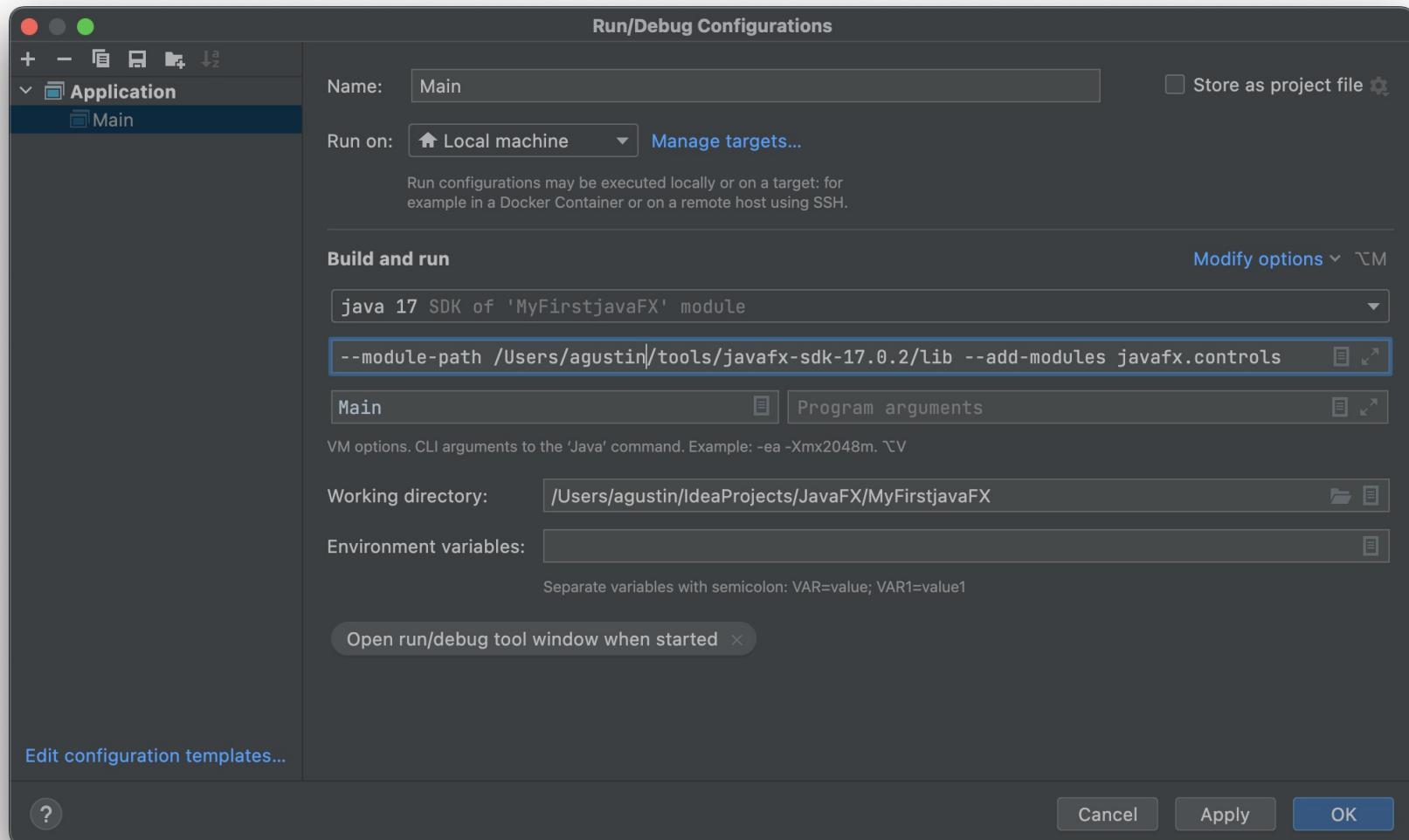
In the "Run" tool window, the configuration is set to run "Main". The output pane shows the following error message:

```
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java -Djava.library.path=/Users/agustin/tools/javafx-sdk-17.0.2/lib -javafx.main-class=Main
Error: JavaFX runtime components are missing, and are required to run this application
```

At the bottom, the status bar indicates: "All files are up-to-date (2 minutes ago)".

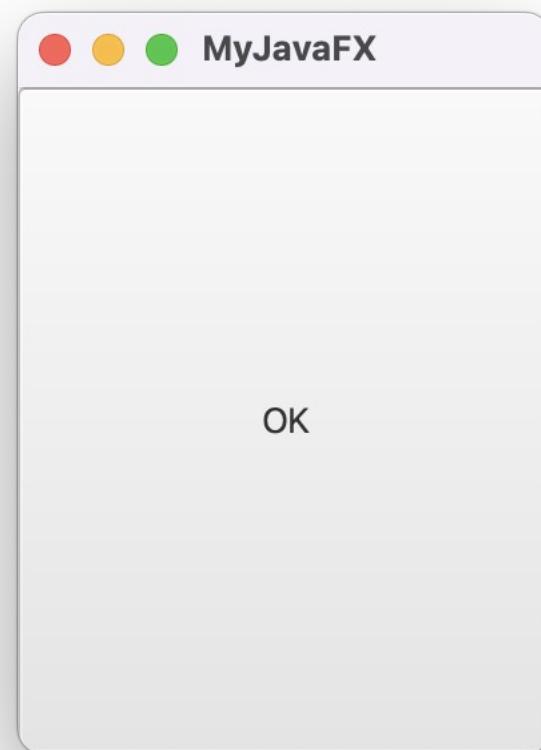
# Edición de la configuración de ejecución

- ❑ Seleccione Run -> Edit Configurations...
- ❑ Seleccione Modify options -> Add VM options. Copie las usadas para compilar en línea de comando.



# Ejecución

- Selecciones Run o de la interfaz el botón simétrico al martillo.



# Volvamos a JavaFX

## Propiedades y Vínculos,

## Eventos de teclado y mouse

# Propiedades y Vínculos (Property, Binding)

- ❑ En JavaFX, las propiedades y vínculos son un mecanismo para expresar relación entre dos variables.
- ❑ Podemos pensar en una Property como un atributo cuyo cambio modifica otros.
- ❑ Cuando un objeto A está vinculado a otro B, A.bind(B), un cambio en B es automáticamente reflejado en A.
- ❑ Bindings son usadas por las GUIs (interfaces gráficas) para mantener el despliegue sincronizado con los datos de la aplicación.
  - Ejemplo: Si un programa cambia el texto de un “label”, la GUI debe reflejar ese cambio en su vista.
- ❑ Este concepto fue usado en el ejemplo [ShowCircleCentered.java](#)
- ❑ Para mantener el círculo centrado en el pane (cristal de la escena), se usó:
  - circle.centerXProperty().bind(pane.widthProperty().divide(2));
  - Así, un cambio en el ancho del pane cambia posición X del centro del círculo.

# Vinculaciones en JavaFX

- ❑ Un cambio hecho a una propiedad de un objeto se reflejará automáticamente en otro objeto.
  - Una interfaz gráfica mantiene su despliegue automáticamente sincronizado con los datos de los objetos gráficos
  - A través de vinculaciones se observa los cambios en los objetos gráficos de los cuales se depende.
- ❑ Veamos un ejemplo en un contexto no gráfico.

# Ejemplo: BindingDemo

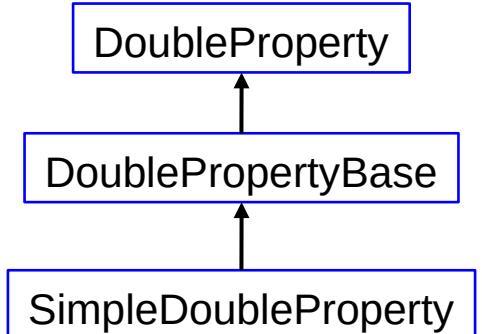
```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class BindingDemo {
    public static void main(String[] args) {
        DoubleProperty d1 = new SimpleDoubleProperty(1.0);
        DoubleProperty d2 = new SimpleDoubleProperty(2.0);
        d1.bind(d2); // d1 value is bound to d2 value.
        System.out.println("d1 is " + d1.getValue()
                           + " and d2 is " + d2.getValue());
        d2.setValue(70.2);
        System.out.println("d1 is " + d1.getValue()
                           + " and d2 is " + d2.getValue());
    }
}
```

La salida es:

d1 is 2.0 and d2 is 2.0

d1 is 70.2 and d2 is 70.2. /\* Notar cambio en d1, aun cuando el  
programa solo cambió d2. \*/



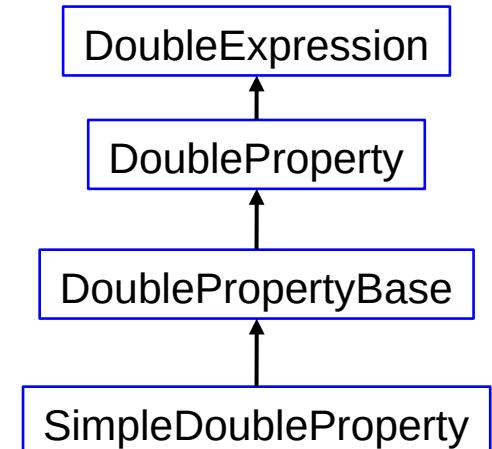
La clase SimpleDoubleProperty contiene un único número double con todo lo necesario para ser vinculable.

# Ejemplo: BindingAddDemo

- La clase DoubleExpresión tiene métodos para sumar, dividir, multiplicar y restar propiedades.

```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.binding.DoubleBinding;

public class BindingAddDemo {
    public static void main(String[] args) {
        DoubleProperty num1 = new SimpleDoubleProperty(3.0);
        DoubleProperty num2 = new SimpleDoubleProperty(7.0);
        DoubleBinding sum = num1.add(num2);
        System.out.println("sum is : "+sum.getValue());
        num1.set(5.0);
        System.out.println("Now sum is : "+sum.getValue());
    }
}
```

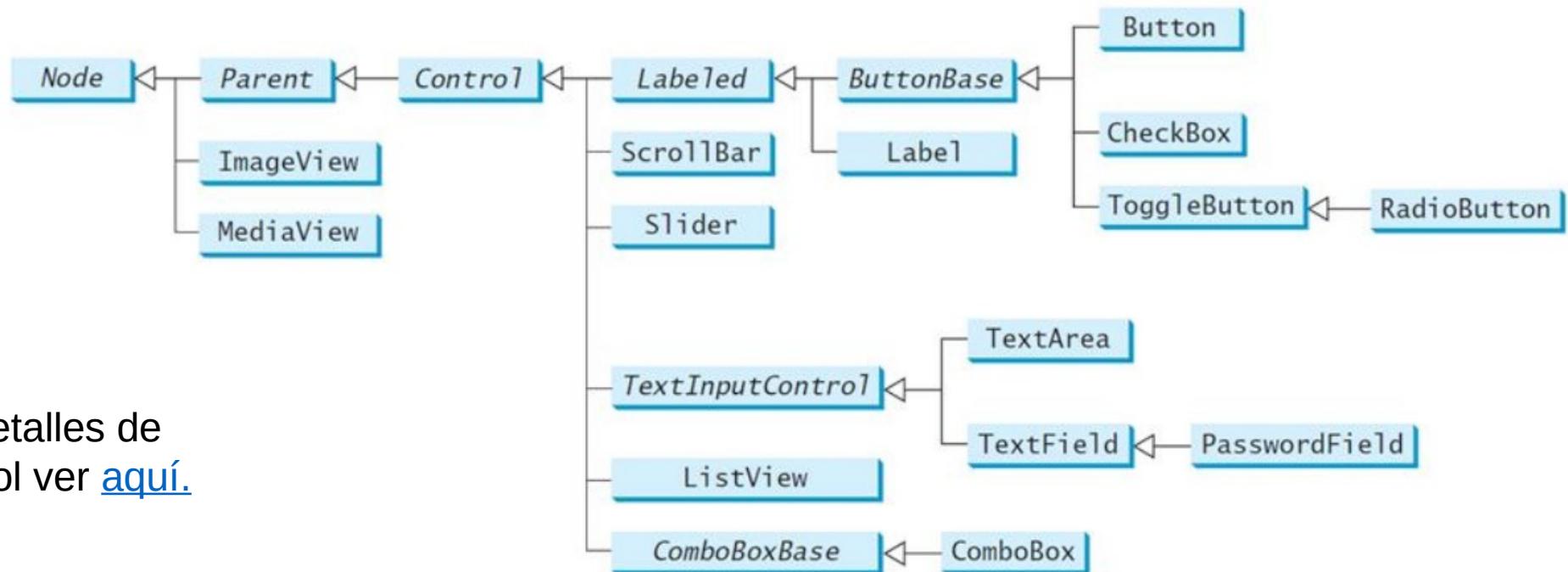


La salida es:  
sum is : 10.0  
Now sum is : 12.0

# Eventos de Teclado y Mouse

# Controles en JavaFX

- La clase Control es la base para todos los nodos en la escena que pueden ser manipulados por el usuario y por ello soportan interacciones con el usuario.



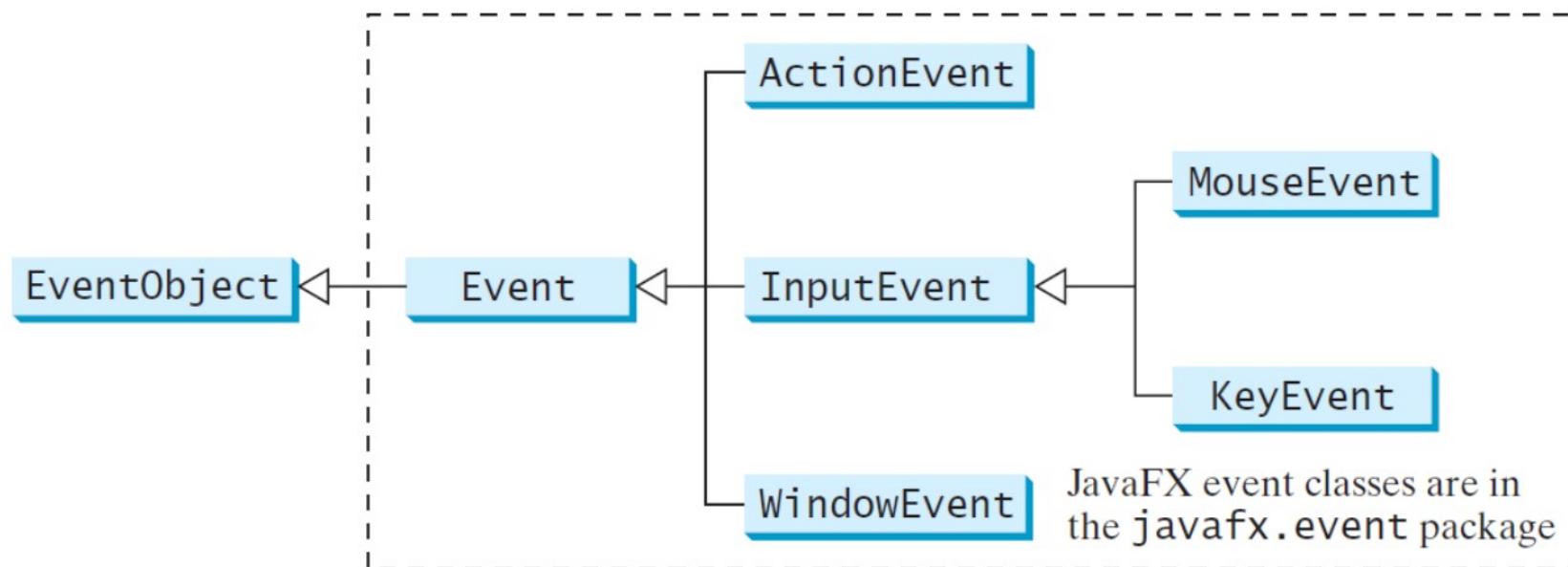
Para ver detalles de cada control ver [aquí](#).

# Programación conducida por eventos (re-visitada)

- ❑ Todos los controles previos pueden detectar diferentes tipos de eventos.
- ❑ Ante la interacción con el usuario, un objeto event es construido y enviado al objeto registrado como manipulador (handler) para ese evento.
- ❑ Los manipuladores de eventos (handlers) son definidos por el programador implementando los métodos de la interfaz asociada al event. Luego el programador debe registrar el handler con objeto gráfico correspondiente (Button, Textfield, ComboBox, etc)

# Posibles Clases para Eventos

- Cuando se invoca un método en respuesta al evento, un objeto evento es pasado. Las clases posibles de eventos son:



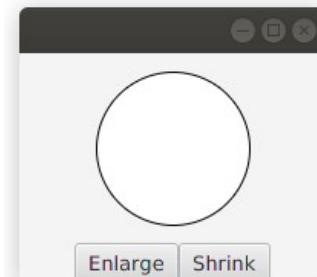
# Algunas Acciones de Usuario y sus Handlers

<i>Acción del usuario</i>	<i>Objeto gráfico</i>	<i>Tipo de evento generado</i>	<i>Registro del handler</i>
Click un botón	Button	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Enter en campo de texto	TextField	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Seleccionar o no	RadioButton	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Seleccionar o no	CheckBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Seleccionar ítem nuevo	ComboBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Presiona mouse	Node, Scene	MouseEvent	setOnMousePressed(EventHandler<MouseEvent>)
Libera mouse	Node, Scene	MouseEvent	setOnMouseReleased(EventHandler<MouseEvent>)
Click mouse	Node, Scene	MouseEvent	setOnMouseClicked(EventHandler<MouseEvent>)
Ingresa mouse	Node, Scene	MouseEvent	setOnMouseEntered(EventHandler<MouseEvent>)
Saca mouse	Node, Scene	MouseEvent	setOnMousePressed(EventHandler<MouseEvent>)
Mueve mouse	Node, Scene	MouseEvent	setOnMouseMoved(EventHandler<MouseEvent>)
Arrastra mouse	Node, Scene	MouseEvent	setOnMouseDragged(EventHandler<MouseEvent>)
Presiona tecla	Node, Scene	keyEvent	setOnKeyPressed(EventHandler<KeyEvent>)
Libera tecla	Node, Scene	keyEvent	setOnKeyReleased(EventHandler<KeyEvent>)
Tipea tecla	Node, Scene	keyEvent	setOnKeyTyped(EventHandler<KeyEvent>)

# Ejemplo 1: Eventos de Control de círculo con dos botones

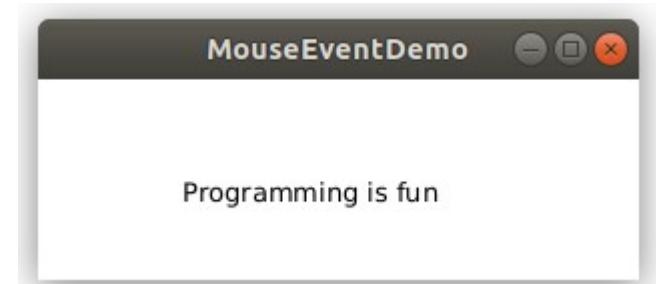
```
public class ControlCircle extends Application {  
    private CirclePane circlePane = new CirclePane();  
  
    @Override  
    public void start(Stage primaryStage) {  
        HBox hBox = new HBox();  
        Button btEnlarge = new Button("Enlarge");  
        Button btShrink = new Button("Shrink");  
        hBox.getChildren().add(btEnlarge);  
        hBox.getChildren().add(btShrink);  
        btEnlarge.setOnAction(new EnlargeHandler()); // using  
        // inner class  
        btShrink.setOnAction(e -> circlePane.shrink()); // Lambda  
        // expression  
        BorderPane borderPane = new BorderPane();  
        borderPane.setCenter(circlePane);  
        borderPane.setBottom(hBox);  
        hBox.setAlignment(Pos.CENTER);  
        Scene scene = new Scene(borderPane, 200, 150);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
  
    class EnlargeHandler implements EventHandler<ActionEvent> {  
        @Override  
        public void handle(ActionEvent e) {  
            circlePane.enlarge();  
        }  
    }  
}
```

```
class CirclePane extends StackPane {  
    private Circle circle = new Circle(50);  
  
    public CirclePane() {  
        getChildren().add(circle);  
        circle.setStroke(Color.BLACK);  
        circle.setFill(Color.WHITE);  
    }  
    public void enlarge() {  
        circle.setRadius(circle.getRadius() + 2);  
    }  
    public void shrink() {  
        circle.setRadius(circle.getRadius() > 2  
            ? circle.getRadius() - 2 :  
            circle.getRadius());  
    }  
}
```



## Ejemplo 2: Eventos de Mouse

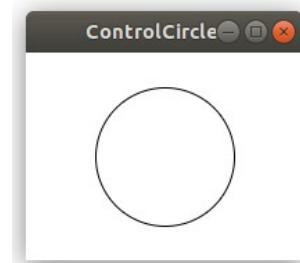
```
public class MouseEventDemo extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
        Pane pane = new Pane(); //no particular layout is used.  
        Text text = new Text(20, 20, "Programming is fun");  
        pane.getChildren().add(text);  
        text.setOnMouseDragged(e -> { // e is the object fired by  
            text.setX(e.getX()); // the event.  
            text.setY(e.getY());  
        });  
        Scene scene = new Scene(pane, 300, 100);  
        primaryStage.setTitle("MouseEventDemo");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
}
```



# Ejemplo 3: Eventos teclado y botones mouse

```
public class ControlCircleWithMouseAndKey extends Application {  
    private CirclePane circlePane = new CirclePane();  
  
    public void start(Stage primaryStage) {  
        Scene scene = new Scene(circlePane, 200, 150);  
        primaryStage.setTitle("ControlCircle");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
        circlePane.setOnMouseClicked(e -> { // mouse click handler  
            if (e.getButton() == MouseButton.PRIMARY) {  
                circlePane.enlarge();  
            } else if (e.getButton() == MouseButton.SECONDARY) {  
                circlePane.shrink();  
            }  
        });  
        scene.setOnKeyPressed(e -> { // Key click handler  
            if (e.getCode() == KeyCode.UP) {  
                circlePane.enlarge();  
            } else if (e.getCode() == KeyCode.DOWN) {  
                circlePane.shrink();  
            }  
        });  
    }  
}
```

```
class CirclePane extends StackPane {  
    private Circle circle = new Circle(50);  
  
    public CirclePane() {  
        getChildren().add(circle);  
        circle.setStroke(Color.BLACK);  
        circle.setFill(Color.WHITE);  
    }  
    public void enlarge() {  
        circle.setRadius(circle.getRadius() + 2);  
    }  
    public void shrink() {  
        circle.setRadius(circle.getRadius() > 2 ?  
                        circle.getRadius() - 2  
                        : circle.getRadius());  
    }  
}
```



# Constantes KeyCode

- ❑ Algunas de las constantes del tipo enumerativo KeyCode son:

Constante	Descripción	Constante	Descripción
<b>HOME</b>	Tecla Home	<b>CONTROL</b>	Tecla Control
<b>END</b>	Tecla End	<b>SHIFT</b>	Tecla Shift
<b>PAGE_UP</b>	Tecla Page Up	<b>BACK_SPACE</b>	Tecla Backspace
<b>PAGE_DOWN</b>	Tecla Page Down	<b>CAPS</b>	Tecla Caps Lock
<b>UP</b>	Tecla up-arrow	<b>NUM_LOCK</b>	Tecla Num Lock
<b>DOWN</b>	Tecla down-arrow	<b>ENTER</b>	Tecla Enter
<b>LEFT</b>	Tecla left-arrow	<b>UNDEFINED</b>	KeyCode desconocido
<b>RIGHT</b>	Tecla right-arrow	<b>F1 a F12</b>	Teclas funciones
<b>ESCAPE</b>	Tecla Esc	<b>0 a 9</b>	Teclas números
<b>TAB</b>	Tecla Tab	<b>A a Z</b>	Teclas letras