Tema 12: Entorno Gráfico:
03 Interacción mediante eventos

Programación

Desarrollo de Aplicaciones Multiplataforma.

Versión: 20240417.2345

Índice de Contenidos

1. Interacción mediante eventos 3
1.1. Eventos 3
1.1.1. Tipos de Evento 3
1.2. Manejadores de eventos 4
1.2.1. Añadir Manejador de Eventos usando SceneBuilder 4
1.2.2. Añadir Manejador de Eventos por código 6
1.3. Ejercicios 12
1.3.1. Ejercicio 01 – Eventos de ratón 12
1.3.2. Ejercicio 02 – Control tamaño de un Círculo 13
1.3.3. Ejercicio 03 – Mover Círculo – Eventos de Teclado 15
1.3.4. Ejercicio 04 – Sumar o Restar – Interactuar con el usuario 16

1. Interacción mediante eventos

Evento = notificación de que algo ha ocurrido.

Cuando un usuario hace clic en un botón, presiona una tecla, mueve el ratón o realiza otras acciones, se generan eventos

Manejador de eventos = método a ejecutar en respuesta a la ocurrencia de ciertos eventos

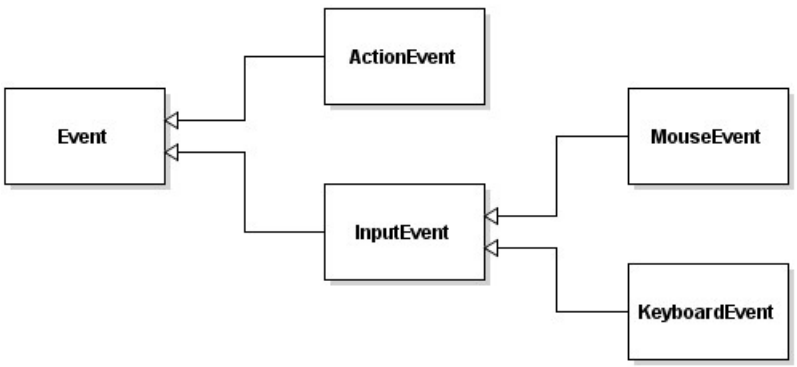
1.1. Eventos

Cualquier evento es una instancia de la clase Event

Los atributos más importantes de la clase Event son:

- eventType: el tipo de evento, un objeto de tipo EventType.
- source: el objeto donde se produce el evento, su fuente (por ejemplo un botón puede ser la fuente de un evento ActionEvent, el cual se dispara al pulsar el botón)

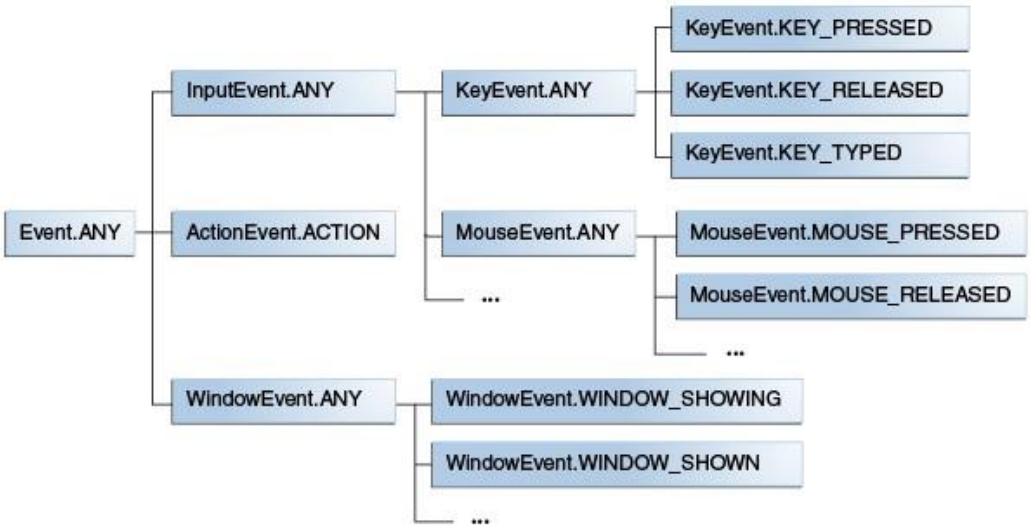
La clase Event tiene varias subclases. Cada subclase puede tener atributos específicos. Por ejemplo, un MouseEvent tiene unas coordenadas x e y que indican en qué píxel se ha producido el evento de ratón.



1.1.1. Tipos de Evento

Para cada subclase de la clase Event se definen una o más instancias de la clase EventType

La clase EventType tiene un atributo denominado superType, de tipo EventType, mediante el cual se establece una jerarquía de objetos, tal y como refleja la siguiente imagen.



1.2. Manejadores de eventos

Los componentes de JavaFX generan eventos al interaccionar con ellos.

Para responder a esos eventos hay que registrar manejadores asociados a algún tipo de evento en concreto:

- Usando scene builder: indicando el fx:ide del nodo y la cabecera del manejador a ejecutar en el tipo de evento a capturar. Después, es necesario actualizar el controlador (Make Controller sobre el FXML)
- Mediante código:
  - Usando el método addEventHandler de la clase Node

```
void addEventHandler(EventType<T> eventType, EventHandler<? super T> eventHandler)
```

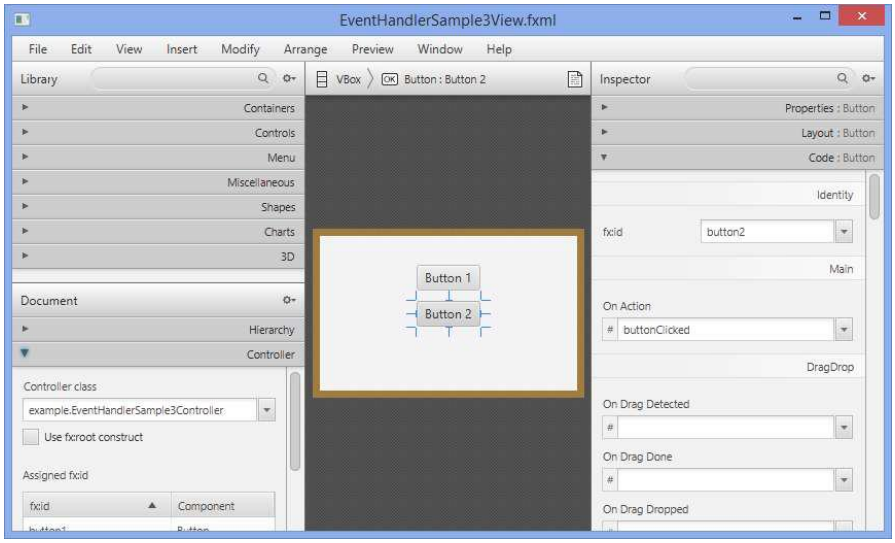
  - O utilizando métodos de conveniencia de la forma

```
setOnEventType(EventHandler<? super T> eventHandler)
```

Un manejador debe implementar la interfaz EventHandler<T extends Event>, la cual declara un único método void handle(T event): método invocado cuando ocurra un evento del tipo para el cual se registra el manejador

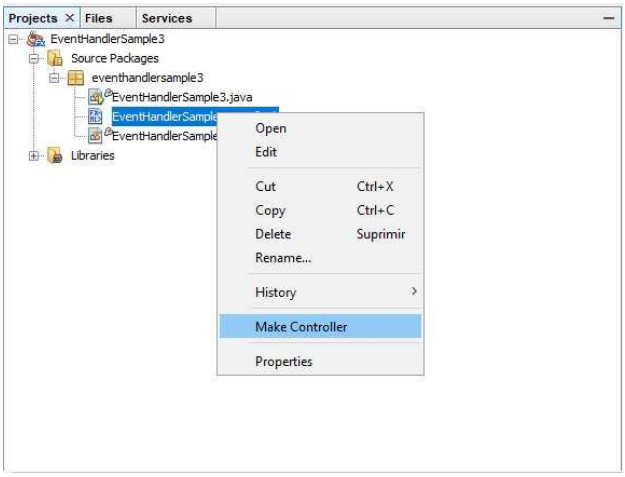
1.2.1. Añadir Manejador de Eventos usando SceneBuilder

añadir id y cabecera del manejador



FXML + Controlador

```
<VBox alignment="CENTER" prefHeight="150.0" prefWidth="250.0" spacing="10.0"
xmlns="http://javafx.com/javafx/8.0.65" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="example.EventHandlerSample3Controller">
    <children>
        <Button fx:id="button1" mnemonicParsing="false" onAction="#buttonClicked"
text="Button 1" />
        <Button fx:id="button2" mnemonicParsing="false" onAction="#buttonClicked"
onMouseEntered="#mouseEntered" onMouseExited="#mouseExited" text="Button 2" />
        <Label fx:id="label">
            <font>
                <Font name="Times New Roman" size="22.0" />
            </font>
        </Label>
    </children>
</VBox>
```




```
public class EventHandlerSample3Controller
{
    @FXML
    private Button button1;
    @FXML
    private Button button2;
    @FXML
    private Label label;
    private static final DropShadow shadow =
new DropShadow();
```

onAction

```
    @FXML
    void buttonClicked(ActionEvent event) {
        String id = ((Node) event.getSource()).getId();
        if (id.equals("button1")) {
            label.setText("Button 1");
        } else {
            label.setText("Button 2");
        }
    }

    @FXML
    void mouseEntered(MouseEvent event) {
        button2.setEffect(shadow);
    }

    @FXML
    void mouseExited(MouseEvent event) {
        button2.setEffect(null);
    }
}
```

1.2.2. Añadir Manejador de Eventos por código

Opciones:

- Métodos de conveniencia: forma abreviada para los eventos más comunes
- addEventHandler: permite añadir manejador a cualquier tipo de evento

En ambos casos, existen formas diferentes y equivalentes de indicar el Código a ejecutar:

- Clases internas
- Clases anónimas
- Funciones lambda → forma abreviada, se escribe menos código
- Referencias a métodos

1.2.2.1. Métodos de conveniencia

- ActionEvent
  - `setOnAction(EventHandler<ActionEvent> value)`
- KeyEvent
  - `setOnKeyTyped(EventHandler<KeyEvent> value)`
  - `setOnKeyPressed(…)`
  - `setOnKeyReleased(…)`
- MouseEvent
  - `setOnMouseClicked(EventHandler<MouseEvent> value)`
  - `setOnMouseEntered(…)`
  - `setOnMouseExited(…)`
  - `setOnMousePressed(…)`

Ejemplo: Hello World con métodos de conveniencia

```
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Hello World");
    Group root = new Group();
    Scene scene = new Scene(root, 300, 250);
    Button btn = new Button();
    btn.setLayoutX(100);
    btn.setLayoutY(80);
    btn.setText("Hello World");
    btn.setOnAction(new EventHandler<ActionEvent>() {

        public void handle(ActionEvent event) {
            System.out.println("Hello World");
        }
    });
    root.getChildren().add(btn);
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

1.2.2.2. Clases internas

Una clase interna, o clase anidada, es una clase definida dentro del ámbito de otra clase.

Las clases internas son útiles para definir clases manejadoras.

Ejemplo de clase interna:

```
// OuterClass.java: inner class demo
public class OuterClass {
  private int data;

  /** A method in the outer class */
  public void m() {
    // Do something
  }

  // An inner class
  class InnerClass {
    /** A method in the inner class */
    public void mi() {
      // Directly reference data and method
      // defined in its outer class
      data++;
      m();
    }
  }
}
```

La clase interna solo es usada por la clase externa y desde ella podemos acceder a los datos y métodos de la clase externa.

Una clase manejador está diseñada para crear un objeto manejador por lo que es apropiado que se defina dentro de la clase principal como una clase interna.

Clases internas

```
@Override
public void start(Stage stage) {
  …
button1.addEventHandler(ActionEvent.ACTION, new
Button1ActionHandler());
  …
}

class Button1ActionHandler implements
EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent event) {
      label.setText("Button 1");
    }
}
```

Añadimos un manejador al evento on action, usando el método AddEventHandler y clases INTERNAS

1.2.2.3. Clases anónimas

Una clase interna anónima es una clase interna sin nombre. Combina la definición de una clase interna y la creación de una instancia de la clase en un solo paso.

Los manejadores de clases internas pueden acortarse utilizando clases internas anónimas.

```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new EnlargeHandler());
}

class EnlargeHandler
    implements EventHandler<ActionEvent> {
  public void handle(ActionEvent e) {
    circlePane.enlarge();
  }
}
```

(a) Inner class Enlargelistener

```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new class EnlargeHandler
      implements EventHandler<ActionEvent>() {
      public void handle(ActionEvent e) {
        circlePane.enlarge();
      }
    });
}
```

(b) Anonymous inner class

Clases anónimas

```
@Override
public void start(Stage stage) {
....
button1.addEventHandler(ActionEvent.ACTION, new
EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
           label.setText("Button 1");
        }
});
…
}
```

Añadimos un manejador al evento on action, usando el método AddEventHandler y clases ANÓNIMAS

1.2.2.4. Expresiones Lambda

Utilizadas para simplificar el código. Pueden verse como una clase anónima con una sintaxis más concisa.

El ejemplo anterior quedaría:

Funciones Lambda

```
@Override
public void start(Stage stage) {
  …

button1.setOnAction((ActionEvent e) -> {
        label.setText("Button 1");
    });
…
}
```

Añadimos un manejador al evento on action, usando el método de CONVENIENCIA setOnAction y una función Lambda

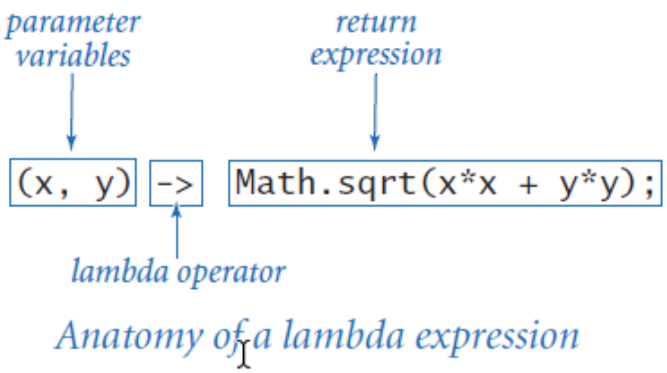
Así que ahora sabemos cómo implementar una clase sin darle un nombre. Pero para clases sencillas, como las que se basan en interfaces con un solo método, incluso una clase interna anónima puede parecer un poco excesiva. Podemos incluso reducir el código un poco más con las expresiones lambda de apoyo de Java.

```java
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;


public class HelloWorldAnonFinal extends Application {

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

    public void start(Stage mainStage) {
        mainStage.setTitle("Hello World Program (AnonFinal)");

        Button btn = new Button();
        btn.setText("Print Hello World!");

        btn.setOnAction(new EventHandler<ActionEvent>() {

            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 300);
        mainStage.setScene(scene);
        mainStage.show();

    }

}
```

```java
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;


public class HelloWorldLambda extends Application {

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

    public void start(Stage mainStage) {
        mainStage.setTitle("Hello World Program (Lambda)");

        Button btn = new Button();
        btn.setText("Print Hello World!");

        btn.setOnAction((ActionEvent event) -> {
                System.out.println("Hello World!");
            }
        );

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 300);
        mainStage.setScene(scene);
        mainStage.show();

    }

}
```

El programa de la derecha utiliza una expresión lambda para reemplazar la clase interna anónima de la izquierda. La expresión lambda es muy similar a la definición de la clase interna anónima, pero aprovecha el hecho de que sólo hay un método abstracto en la interfaz del manejador de eventos que puede hacer las cosas más concisas.

Como podemos ver, podríamos omitir el nombre del método porque sólo hay una opción en el manejador de la interfaz de eventos. Hay incluso más cosas que podemos omitir como el modificador public void y el tipo de retorno. Incluso podemos omitir el tipo de parámetro formal ya que Java puede determinar eso desde la interfaz, todo lo que acabamos necesitando es un nombre de parámetro formal, el símbolo lambda que es un guión y un signo mayor que.

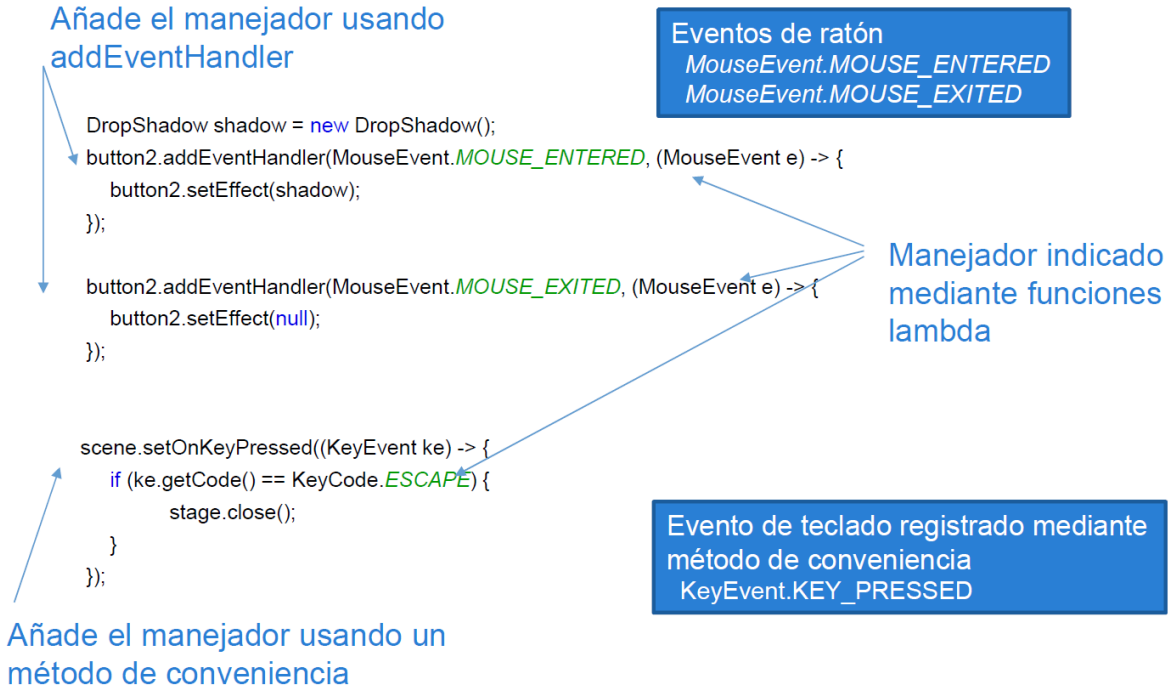


Anatomy of a lambda expression

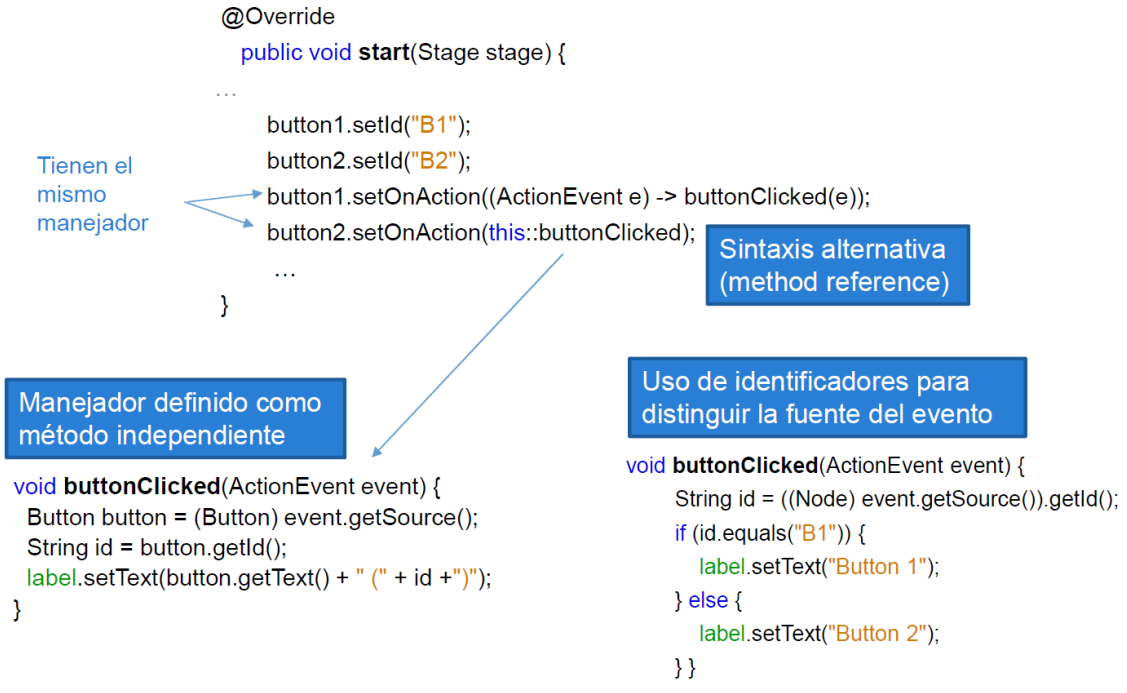
Y después de eso está el cuerpo del método, tenemos sólo una línea de código aquí en el cuerpo del método, así que las llaves podrían incluso ser omitidas en este caso, para que todo sea una línea de código.

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorldLambda extends Application {

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

    public void start(Stage mainStage) {
        mainStage.setTitle("Hello World Program (Lambda)");

        Button btn = new Button();
        btn.setText("Print Hello World!");
        btn.setOnAction(event -> System.out.println("Hello World!"));
        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 300);
        mainStage.setScene(scene);
        mainStage.show();
    }
}
```

1.2.2.5. Un ejemplo con todo lo anterior:



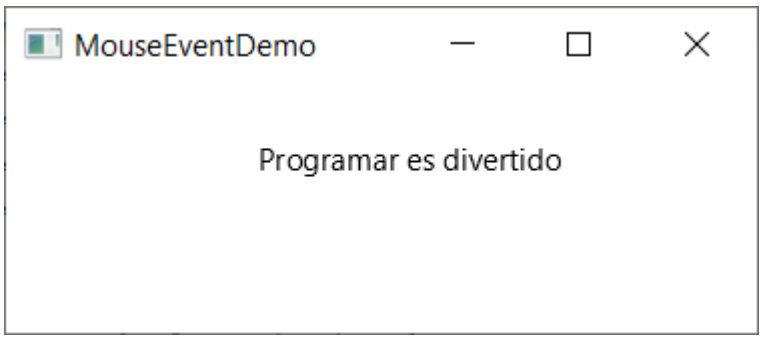
1.2.2.6. Referencia a métodos



1.3. Ejercicios

1.3.1. Ejercicio 01 – Eventos de ratón

Crear una aplicación como la siguiente vista:



De manera que podamos arrastrar el texto por la ventana.

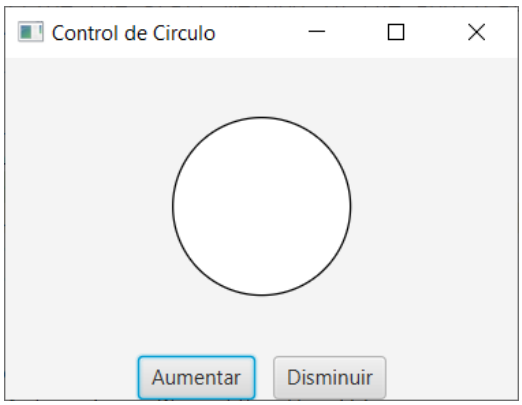
Para ello debemos crear un manejador para el evento MouseDragged (setOnMouseDragged)

1.3.2. Ejercicio 02 – Control tamaño de un Círculo

El objetivo del proyecto es trabajar con eventos: de botones, teclado y ratón.

Crear un proyecto JavaFX cumpliendo lo siguiente:

Una ventana con un círculo, de manera que aumente (o disminuya) su radio en respuesta a los eventos producidos al pulsar los botones y al clic del ratón (izquierdo aumentar, derecho disminuir)



Código de la vista:

```
public class ControlCirculo extends Application {
  @Override // Sobrescribimos el método start de la clase Application
  public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
    Circle circle = new Circle(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);
    pane.getChildren().add(circle);

    HBox hBox = new HBox();
    hBox.setSpacing(10);
    hBox.setAlignment(Pos.CENTER);
    Button btAumentar = new Button("Aumentar");
    Button btDisminuir = new Button("Disminuir");
    hBox.getChildren().add(btAumentar);
    hBox.getChildren().add(btDisminuir);

    BorderPane borderPane = new BorderPane();
    borderPane.setCenter(pane);
    borderPane.setBottom(hBox);
    BorderPane.setAlignment(hBox, Pos.CENTER);

    // Creamos una scene y la situamos en el stage
    Scene scene = new Scene(borderPane, 200, 150);
    primaryStage.setTitle("Control de Círculo"); // Establecemos el título de la ventana
    primaryStage.setScene(scene); // Situamos la escena (scene) en la ventana (stage)
    primaryStage.show(); // Mostramos stage
  }
```

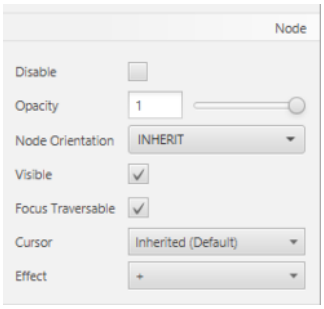
```
  /**
   *
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

1.3.3. Ejercicio 03 – Mover Círculo – Eventos de Teclado

El objetivo del proyecto es trabajar con eventos de teclado.

Crear un proyecto JavaFX cumpliendo lo siguiente:

- Añadir al grafo de escena un gridpane de 5x5 celdas.
- Añadir un círculo en el centro del grid. Indicar su fila y columna en sus propiedades
- Añadir la gestión de eventos para poder mover el botón mediante las teclas de scroll.
- Haz que el estilo de la ventana sea TRANSPARENT
- Añade un método de conveniencia para que los usuarios puedan salir de la aplicación pulsando la tecla escape
- El círculo que controlar debe tener la propiedad Focus Traversable seleccionada



Métodos estáticos de GridPane útiles:

- GridPane.getRowIndex(miNodo): devuelve fila en la que se encuentra el objeto pasado por parámetro
- GridPane.getColumnIndex(miNodo): devuelve columna del objeto pasado por parámetro

Métodos para cambiar a un nodo de fila o columna:

- miGrid.setRowIndex(miNodo, numFila)
- miGrid.setColumnIndex(miNodo, numColumna)

Obtener el código del evento creado (por ejemplo qué tecla fue pulsada):

- miEvento.getCode()

Códigos de las teclas:

- KeyCode.RIGHT, KeyCode.LEFT, …

Métodos de Stage útiles:

- miStage.initStyle(StageStyle.TRANSPARENT): estilo transparente, sin iconos ni borde
- miStage.close() : Cierra la Ventana, y la aplicación si es la única que queda

1.3.4. Ejercicio 04 – Sumar o Restar – Interactuar con el usuario

Completar la aplicación de la imagen, con los siguientes requisitos:

- 1. La etiqueta grande centrada muestra un número, al que se le pueden sumar 1, 5 o 10 unidades con los botones
- 2. También se puede sumar una cantidad arbitraria (TextField Valor)
- 3. Al seleccionar el checkbox, en vez de sumar se resta. Además, mientras que esté marcado, la etiqueta de abajo aparece (se oculta en caso contrario).
- 4. La caja de texto permite introducir un número. Al pulsar el botón “Suma” se suma (o resta) al valor de arriba.