



## P2. INTERACCIÓN CON EL USUARIO

---

Interfaces Persona Computador

Depto. Sistemas Informáticos y Computación

UPV

# Índice

- Introducción a los eventos en JavaFX (*Event Handlers*)
- Ejercicio

# Interacción mediante eventos

- Los **eventos** se utilizan para notificar a su aplicación las acciones realizadas por el usuario y permitir que la aplicación responda al evento.
- La plataforma JavaFX proporciona la estructura para capturar un evento, dirigir el evento a su destino y permitir que la aplicación maneje el evento según sea necesario



# Interacción mediante eventos

1. El programador **codifica** métodos (manejador de eventos) para responder a los eventos.
2. El programador **registra** los métodos sobre un objeto de la interfaz.

El usuario hace clic con el ratón sobre un botón, esto es un evento.

1. La plataforma JavaFX crea un objeto `ActionEvent`.
2. La plataforma busca un método registrado y lo llama usando el `ActionEvent` como parámetro



# ¿Que tengo que saber?

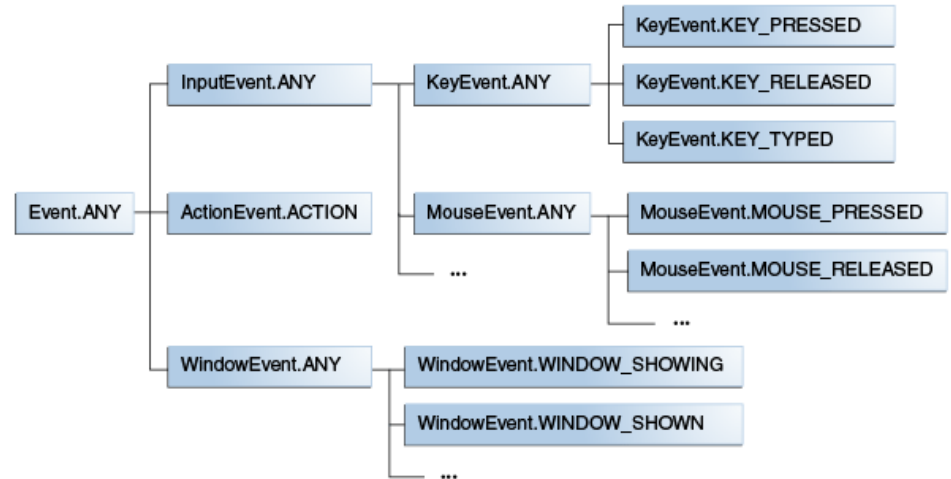
1. Tipos de eventos hay en JavaFX
2. Acciones del usuario que causan estos eventos
3. Cómo codifico un manejador de eventos.
4. Cómo registro un manejador de eventos sobre un objeto de la interfaz

# Eventos

- Cualquier evento es una instancia de la clase *Event*
- **Evento del ratón**: ocurre cuando se utiliza el ratón. Acciones: presionar botón, soltar botón, pulsar, mover el ratón, arrastrar, entrar sobre un nodo, salir sobre un nodo. *MouseEvent*
- **Evento teclado**: ocurre cuando se utiliza el teclado. Acciones: presiona tecla, libera tecla, tecla pulsada, tecla pulsada y mantenida. *KeyEvent*
- **Evento ventana**: ocurre sobre la ventana. Acciones: mostrar ventana, minimizar ventana, cerrar ventana, etc. *WindowEvent*
- **Evento genérico**: es un evento lógico que puede estar generado por el ratón o el teclado. Se genera junto a eventos anteriores. *ActionEvent*

# Eventos

- Atributos incluidos en la clase *Event*
- **eventType**: tipo de evento



- **source**: el objeto donde se produce el evento, un nodo puede ser fuente de varios tipos de eventos.
- **otros**: posición del ratón, botón del ratón pulsado, tecla pulsada, tecla secundaria pulsada, etc.

Node	EventType
Button	ActionEvent
TextField	ActionEvent, KeyEvent
Any kind of Node	MouseEvent, ...
Scene	MouseEvent, KeyEvent,...
Stage	WindowEvent

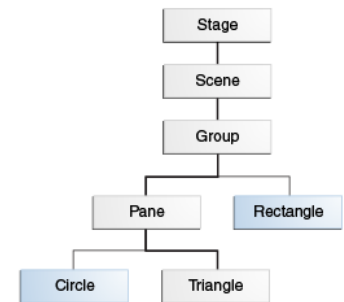
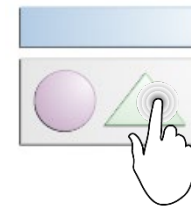
# La plataforma procesa el Evento

## 1. selección del **source**: (IMPORTANTE)

- **KeyEvent**: el evento es dirigido al nodo que tiene el foco (**focused**)
- **MouseEvent**: el evento es dirigido al nodo que esta debajo del puntero del ratón

## 2. construcción de la ruta:

- Se traza una ruta por el grafo de escena desde el Stage al nodo source.

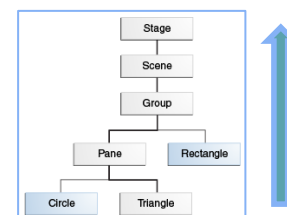
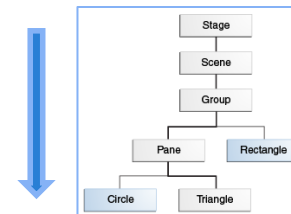


## 3. captura del evento:

- El evento es lanzado desde el Stage hasta el source. Si hay **filtros** se ejecutan.

## 4. propagación del evento:

- El evento llega al source y si tiene **manejador** se ejecuta (y consume). Si no tiene, recorre el camino inverso y si hay **manejadores** los ejecuta.

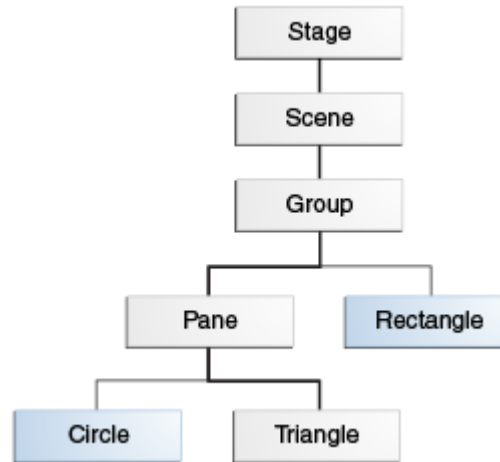
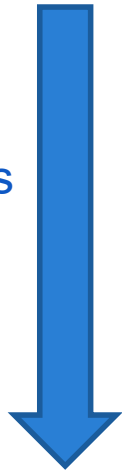




# Captura y propagación

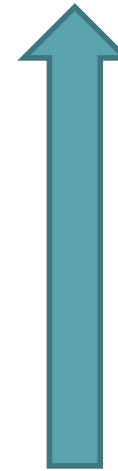
Captura del evento

Se ejecutan **filtros**  
(no los veremos)



Propagación

Se ejecutan **manejadores**



**Event** tiene una propiedad booleana **“consumed”**, en el momento que se le asigna valor true ya no se ejecuta otro método en el camino

Para saber más: <http://docs.oracle.com/javase/8/javafx/events-tutorial/events.htm>

# Manejadores de eventos :

- Un manejador de evento es un método con la interfaz `EventHandler<T>` La cabecera del método es:

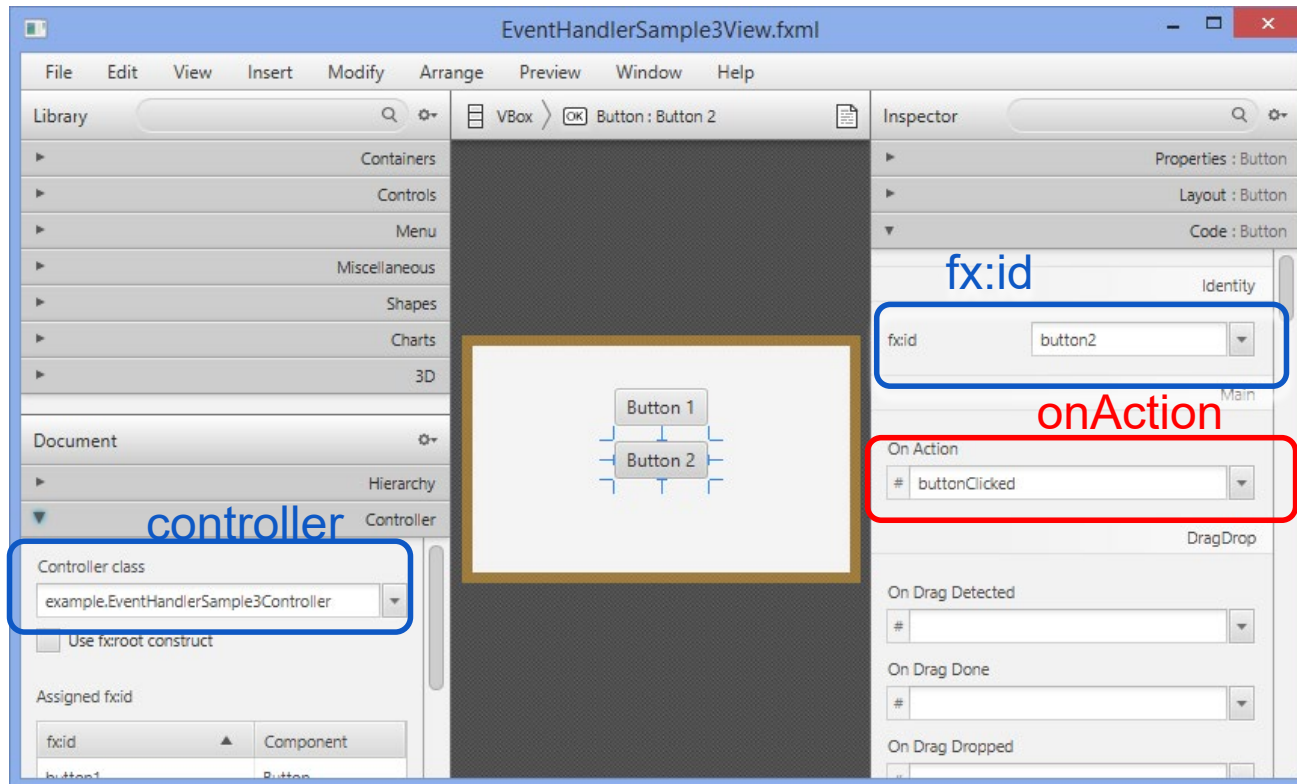
```
void handle(T event) {  
    // aquí hay que codificar la acción  
}
```

```
private void pulsadoIniciar(ActionEvent event) {  
    mensaje_usuario.setText("Bienvenido " + texto_usuario.getText());  
}
```

```
private void canviarCursor(MouseEvent event) {  
    miBola.setCursor(Cursor.HAND);  
}
```

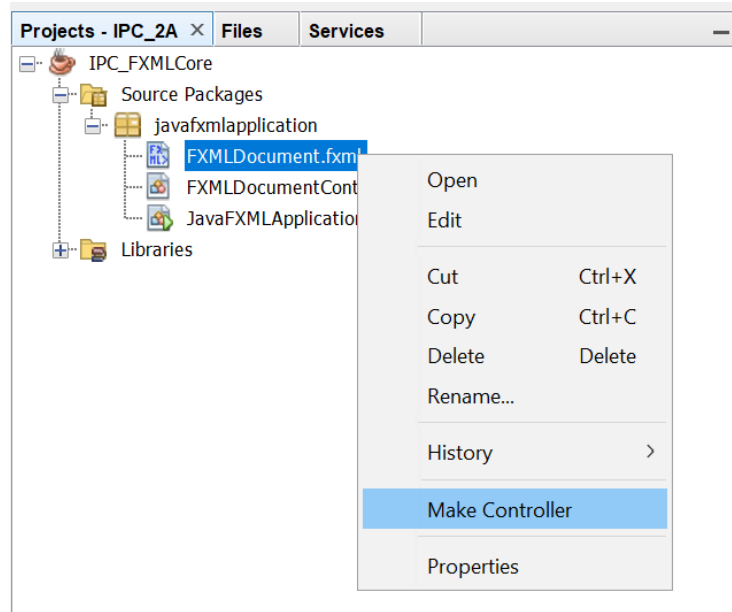
```
private void moverBOLA(KeyEvent event) {  
    System.out.println(event.getCode());  
    KeyCode teclaPulsada = event.getCode();
```

# Registro de un manejador SceneBuilder:



```
@FXML
private void buttonClicked(ActionEvent event) {
    System.out.println("Button 2 clicked!");
}
```

# Sincronizar FXML y la clase Controladora



**Recuerda!!!**, cada vez que modificas un valor relacionado con el código en el fichero FXML es necesario:

- 1- **Salvar** el fichero FXML
- 2- Invocar la opción de menú **Make Controller**

# Registro de un manejador por código

1. *addEventHandler( , )* ( no lo vamos a ver en las prácticas)

2. Métodos de conveniencia:

Los nodos tienen un método específico por cada uno de los tipos de eventos que pueden registrar, la sintaxis es:

**setOn**"Tipo de evento" ( **EventHandler** );

```
public void initialize(URL url, ResourceBundle rb) {  
    // TODO  
    buttonClickCodigo.setOnAction(this::handleButton);  
}  
  
private void handleButton(ActionEvent event) {  
    labelMessage.setText("Hello, this is your first JavaFX project - IPC");  
}
```

*onAction*

# Registro de un manejador por código

```
public void initialize(URL url, ResourceBundle rb) {  
    // TODO  
    buttonClickCodigo.setOnKeyPressed(this::handleButtonTyped);  
}  
private void handleButtonTyped(KeyEvent event) {  
    System.out.println("you type: " + event.getCode());  
}
```

onKeyPressed

```
public void initialize(URL url, ResourceBundle rb) {  
    // TODO  
    buttonClickCodigo.setOnMousePressed(this::handleButtonClick);  
}  
private void handleButtonClick(MouseEvent event) {  
    System.out.println("you Click: " + ((Button) event.getSource()).getText());  
}
```

onMousePressed

```
stage.show();  
stage.setOnCloseRequest(this::handleStageClosed);  
}  
private void handleStageClosed(WindowEvent event) {  
    System.out.println("you clouse the Stage");  
}
```

onCloseRequest

# Registro de un manejador por código

Cuando no necesitamos reutilizar un método, es decir no necesitamos llamar al método varias veces, Java permite crea un método sin nombre: **función LAMBDA**


```
( parámetros ) -> {  
    código del método  
}
```

```
buttonClickCodigo.setAction((event) -> {  
    System.out.println("ActionEvent on: " + ((Button) event.getSource()).getText());  
})  
);
```

No es necesario indicar los tipos, el compilador los detecta automáticamente

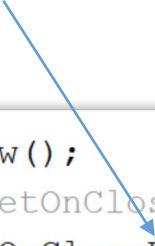
# Registro de un manejador por código

Registro método de  
conveniencia




```
buttonClickCodigo.setOnKeyTyped((event) -> {  
    System.out.println("you have typed: " + event.getCode());  
});
```

Registro método de  
conveniencia



Manejador mediante  
funciones lambda



```
// stage.show();  
stage.setOnCloseRequest(this::handleStageClosed);  
stage.setOnCloseRequest((event) -> {  
    System.out.println("you clouse the Stage");  
});  
}
```



**Ejercicio:** (El objetivo del proyecto es trabajar primero con eventos de teclado y después con eventos del ratón)

- Partir del proyecto JavaFX de base
- 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 (**columna 2, fila 2**)

1. Añadir la gestión de eventos para poder mover el círculo mediante las teclas de dirección.
2. Añadir la gestión de eventos para mover el círculo a la celda pulsada con el ratón
3. Añadir la gestión de eventos para arrastrar y soltar el círculo en una celda



# Manejador de eventos KeyEvent

@FXML

```
public void handleKeyPressed(KeyEvent event) {  
    if (event.getCode() == KeyCode.UP) {  
        // do some actions  
    }  
}
```

Mediante el método `getCode()` obtenemos el código de la tecla pulsada

1. Añadir la gestión de eventos para poder mover el círculo mediante las teclas de dirección.

¿Quién debe de ser el nodo SOURCE?

KeyCode es un tipo enumerado con todas la teclas:

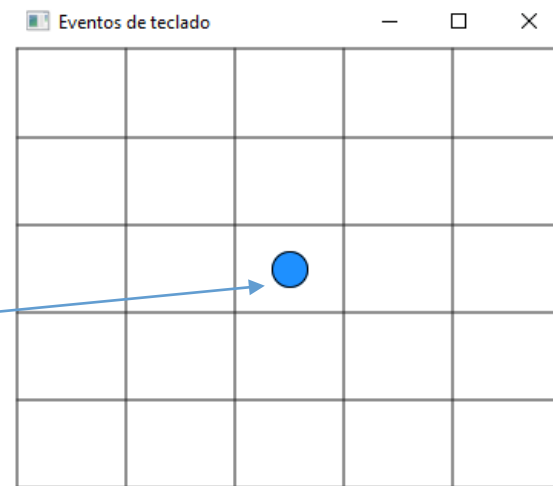
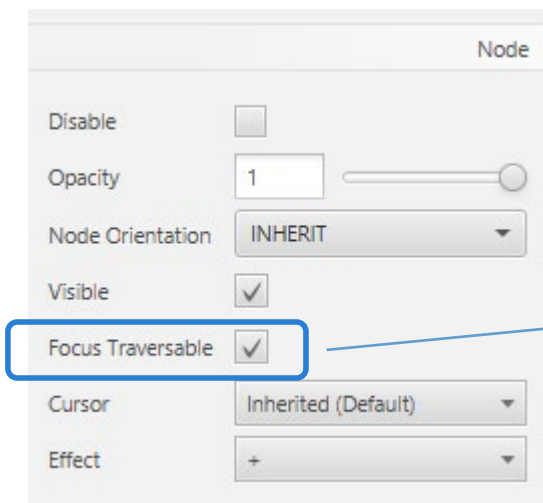
`KeyCode.ENTER`

.....  
`KeyCode.UP`  
`KeyCode.DOWN`  
`keyCode.LEFT`  
`KeyCode.RIGHT`  
.....

# KeyEvent, source es el nodo con el foco

1. Añadir la gestión de eventos para poder mover el círculo mediante las teclas de dirección.

- Para que el círculo pueda recibir eventos del teclado debe de poder recibir el FOCO



# GridPane, métodos útiles para mover nodos

```
// recupera la fila en la que se encuentra el nodo
```

```
int row = GridPane.getRowIndex(nodo);
```

```
// recupera la columna en la que se encuentra el nodo
```

```
int column = GridPane.getColumnIndex(nodo);
```

```
// cambia la fila en la que se encuentra el nodo
```

```
GridPane.setRowIndex(nodo , row);
```

```
// cambia la columna en la que se encuentra el nodo
```

```
GridPane.setColumnIndex(nodo , column);
```

```
// cambia la columna y fila en la que se encuentra el nodo
```

```
GridPane.setConstraints(nodo, column, row);
```

```
// añade el nodo en la columna y fila
```

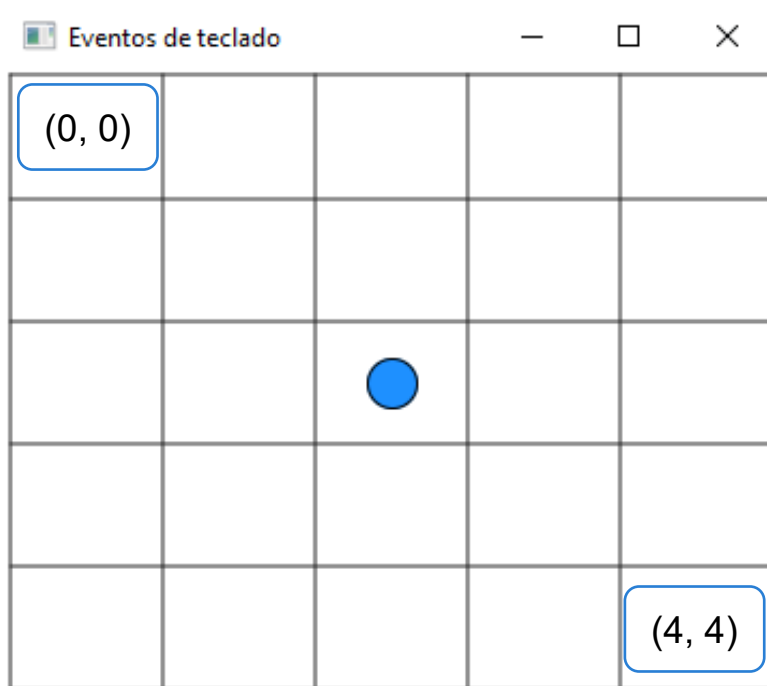
```
myGridPane.add(nodo, column, row);
```

1. Añadir la gestión de eventos para poder mover el círculo mediante las teclas de dirección.

# GridPane, coordenadas

- Las columnas y filas empiezan por 0 en la parte superior izquierda
- Si se mueve un nodo a una fila/columna negativa dará un error
- Si no existe la columna/fila, la creará.

1. Añadir la gestión de eventos para poder mover el círculo mediante las teclas de dirección.



// normalizar cualquier fila

```
public int rowNorm(GridPane grid, int row) {  
    int rowCount = grid.getRowCount();  
    return ( row + rowCount) % rowCount;  
}
```

// normalizar cualquier columna

```
public int columnNorm(GridPane grid, int column) {  
    int columnCount = grid.getColumnCount();  
    return ( column + columnCount) % columnCount;  
}
```

Estas funciones están en  
la clase Utils

# Manejador de eventos MouseEvent

@FXML

```
public void handleMousePressed(MouseEvent event) {  
    double x = event.getSceneX();  
    double y = event.getSceneY();  
    .....  
    // dejar el circulo en la celda adecuada  
    .....  
}
```

2. Añadir la gestión de eventos para mover el círculo a la celda pulsada con el ratón

¿nodo SOURCE?

Mediante los métodos getSceneX(), getSceneY(), obtenemos las coordenadas X, Y del ratón

Calcular la **columna** a la que corresponde una coordenada **x**:

- Dividir la coordenada entre el ancho de una columna y quedarnos con la parte entera.
- El ancho de la columna es el ancho del grid dividido entre el número de columnas.

El proceso es similar para la fila.

```
public int columnCalc(GridPane grid, int x) {  
    int celdaWidth = (int)grid.getWidth() / grid.getColumnCount();  
    return (int) ( x / celdaWidth);  
}
```

```
public int rowCalc(GridPane grid, int y) {  
    int celdaHeigth = (int)grid.getHeight() / grid.getRowCount();  
    return (int) ( y / celdaHeigth);  
}
```

Estas funciones están en la clase Utils

# Manejador de eventos MouseEvent

3. la gestión de eventos para arrastrar y soltar el círculo en una celda

- JavaFX permite pintar objetos en otra posición a la que le corresponde, mediante los métodos `setTranslateX(x_inc)`, `setTranslateY(y_inc)`
- **¡ATENCIÓN!!!** mediante estos métodos:
  - Podemos **pintar un objeto fuera de la celda** que le corresponde en el GridPane,
  - **No cambiamos** el objeto de **celda**.
- Para repintar el círculo al mover el ratón:
  - calcular la **x\_inc** como la diferencia entre el valor de la coordenada un momento determinado (MouseDragged) y el valor cuando empieza el movimiento (MousePressed)
  - Lo mismo para **y\_inc**

$x\_inc = (X\_actual\_raton - X\_inicial\_raton) ; y\_inc = (Y\_actual\_raton - Y\_inicial\_raton)$

@FXML

```
public void handleMousePressed(MouseEvent event) {
```

```
    X_ini = event.getSceneX();
```

```
    Y_ini = event.getSceneY();
```

```
}
```

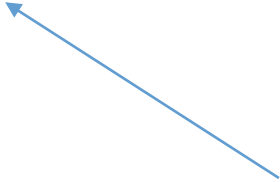
Mediante los métodos `getSceneX()`, `getSceneY()`, obtenemos las coordenadas X, Y del ratón

# Manejador de eventos MouseEvent

3. la gestión de eventos para arrastrar y soltar el circulo en una celda

@FXML

```
public void handleMouseDragged(MouseEvent event) {  
    miCirculo.setTranslateX(event.getSceneX() - X_inicial);  
    miCirculo.setTranslateY(event.getSceneY() - Y_inicial);  
}
```



Mediante los métodos  
setTranslateX(), setTranslateY(),  
modificamos el lugar donde se  
pinta el objeto



# Manejador de eventos MouseEvent

3. la gestión de eventos para arrastrar y soltar el círculo en una celda

@FXML

```
public void handleMouseReleased(MouseEvent event) {  
    miCirculo.setTranslateX(0);  
    miCirculo.setTranslateY(0);  
    // dejar el circulo en la celda adecuada  
    .....  
    event.consume();  
}
```

Mediante el método `consume()`, conseguimos que este evento no se aplique sobre los objetos anteriores en el grafo de escena

# Ejercicio: (AMPLIACIÓN)

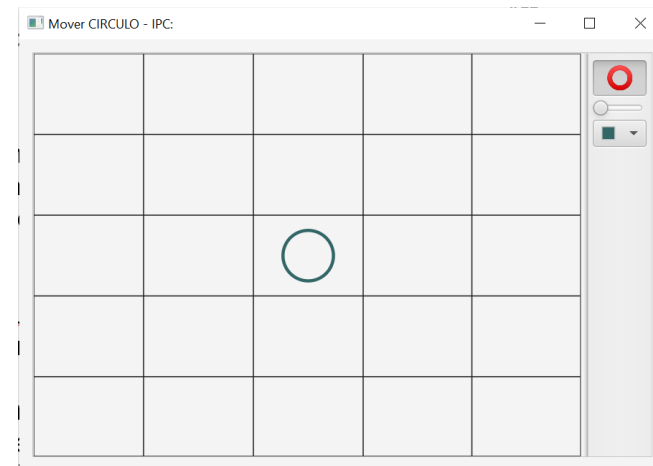
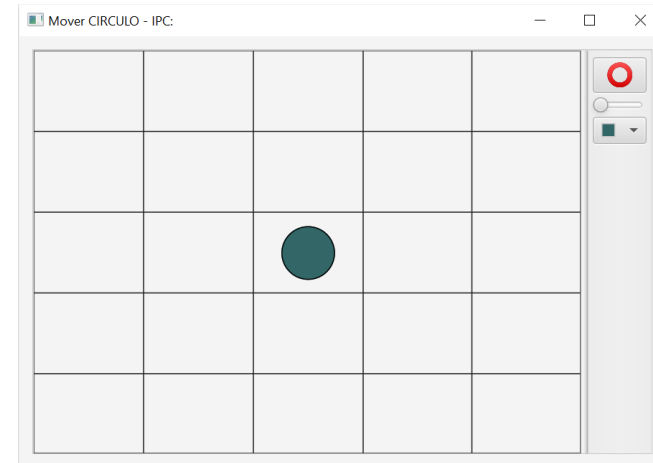
1. Añadir un contenedor del tipo SplitPane vertical
  - Eliminar los pane que se generan por defecto
2. Añadir en la parte de la derecha del SplitPane un VBox o un ToolBar.
  - Modificar orientation= VERTICAL
3. Añadir al ToolBar un ToggleButton con el que gestionar que el círculo este relleno o no. Tamaño 50x25

```
if (miboton. isSelected()) {  
    micirculo. setFill( Color.TRANSPARENT);  
    micirculo. setStrok( micolorPicker. getValue();  
} else { // a la inversa  
}
```

4. Añadir al ToolBar un slider con el que gestionar el tamaño del círculo. Tamaño 50x25
5. Añadir al ToolBar un ColorPicker con el que cambiar el color del círculo. Tamaño 50x25  

```
micirculo. setFill( colorPicker. getValue();
```

Para que los eventos del teclado lleguen solo al círculo es necesario que todos los nodos tengan la propiedad **Focus Traversable** desmarcada



# Referencias

- Tutorial Oracle: Handling JavaFX Events  
<http://docs.oracle.com/javafx/2/events/jfxpub-events.htm>
- API JavaFX 11: <https://openjfx.io/javadoc/17/>
- JavaFX 8 Event Handling Examples:  
<http://code.makery.ch/blog/javafx-8-event-handling-examples/>
- Cálculo Lambda en Java: Raoul-Gabriel Urma, Mario Fusco, and Alan Mycroft, *Java 8 in Action*  
*Lambdas, streams, and functional-style programming*