P2. INTERACCIÓN CON EL USUARIO

Interfaces Persona Computador

Depto. Sistemas Informáticos y Computación

UPV

Índice

- Introducción a los eventos en JavaFX (*Event Handlers*)
- Métodos de conveniencia
- Eventos y manejadores en FXML (SceneBuilder)
- Patrón observador (*Listeners*)
- Propiedades
- Enlace (*Binding*)

Introducción

- Botones y menús
- Selectores, interruptores, barras de desplazamiento, etc.
- Selección de elementos en listas, tablas, etc.
- Gestos en un dispositivo táctil
- …

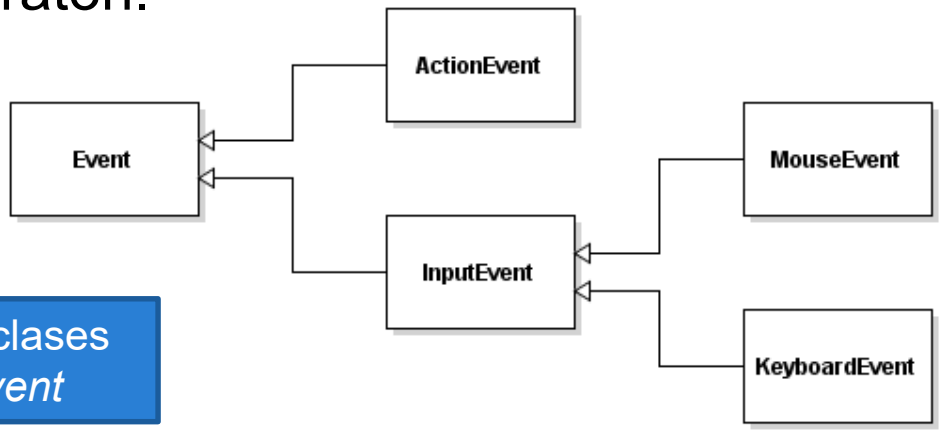
¿Qué tienen en común todos estos mecanismos de interacción?

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

Eventos

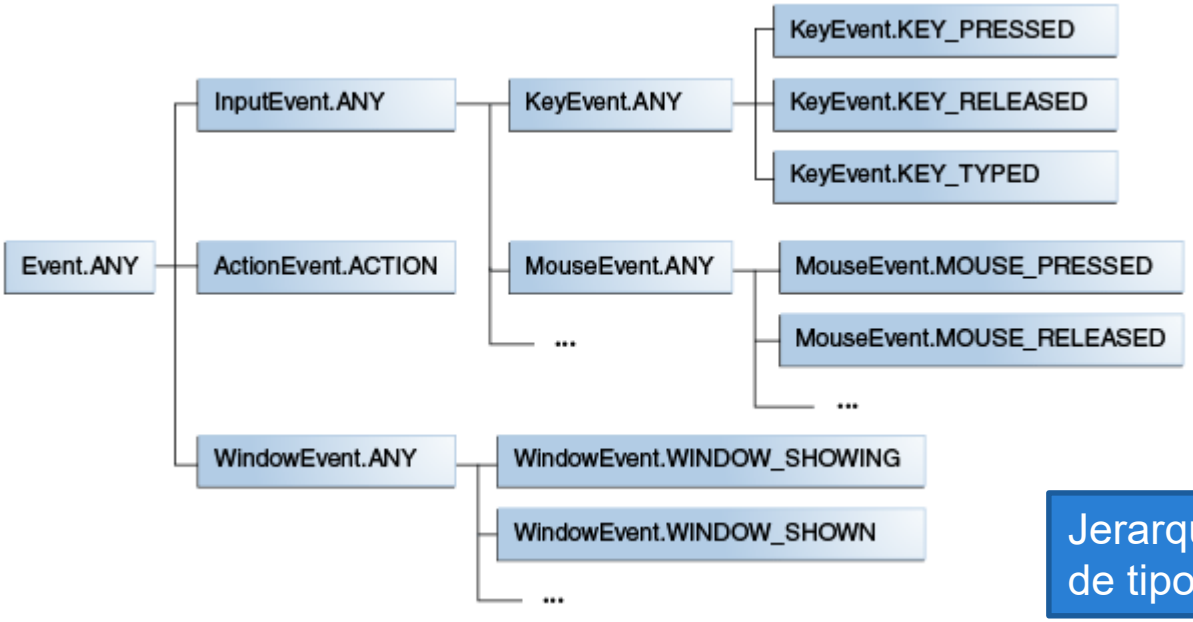
- Cualquier evento es una instancia de la clase *Event*
- Los atributos más importantes de la clase *Event* son
  - ***eventType***: el tipo de evento, un objeto de tipo *EventType*.
  - ***source***: el objeto donde se produce el evento, su fuente (por ejemplo un botón puede ser la fuente de un evento *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 pixel se ha producido el evento de ratón.



Algunas de las clases derivadas de *Event*

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.



Jerarquía de objetos de tipo *EventType*

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.
  - Mediante el método *addEventHandler* de la clase *Node*

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

  - O mediante *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

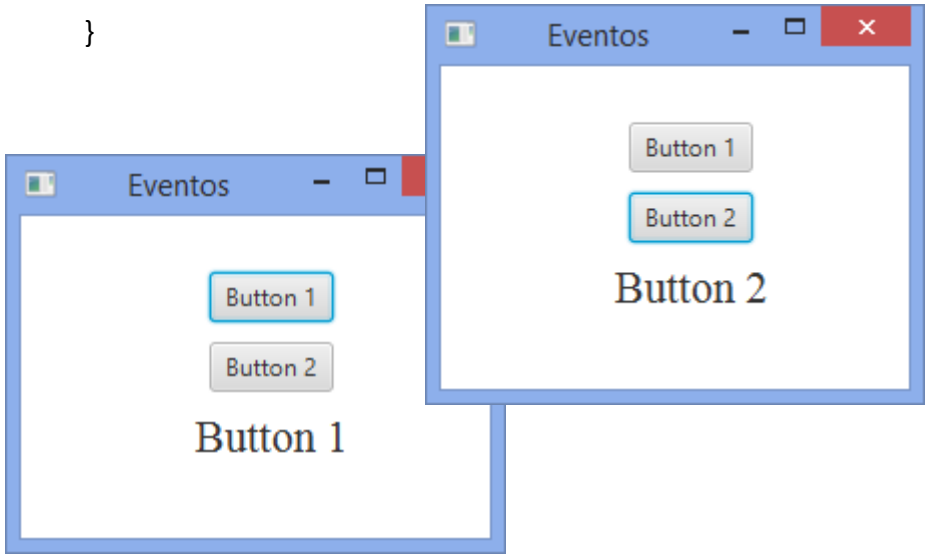
Métodos de conveniencia

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

Para saber más: http://docs.oracle.com/javafx/2/events/convenience_methods.htm

Ej.1 Controlador sin manejadores

```
public class EventHandlerSample1 extends Application {
   private Label label;

   @Override
   public void start(Stage stage) {
      // Creamos los controles
      label = new Label();
      label.setFont(Font.font("Times New Roman", 22));
      Button button1 = new Button("Button 1");
      Button button2 = new Button("Button 2");

// Creamos contenedor (VBox) y añadimos controles
      VBox vbox = new VBox();
      vbox.setAlignment(Pos.CENTER);
      vbox.setSpacing(10);
      vbox.getChildren().add(button1);
      vbox.getChildren().add(button2);
      vbox.getChildren().add(label);

// Creamos escena con el vbox como nodo raíz
      Scene scene = new Scene(vbox, 250, 200);

// Establecemos propiedades del stage
      stage.setTitle("Eventos");
// Establecemos la escena y mostramos stage
      stage.setScene(scene);
      stage.show();
   }

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



Añadiendo manejadores

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");
      }
}
```

Clases anónimas

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

Funciones Lambda

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

Ejemplo extendido

Eventos de ratón
MouseEvent.MOUSE_ENTERED
MouseEvent.MOUSE_EXITED

```
DropShadow shadow = new DropShadow();
button2.addEventHandler(MouseEvent.MOUSE_ENTERED, (MouseEvent e) -> {
    button2.setEffect(shadow);
});

button2.addEventHandler(MouseEvent.MOUSE_EXITED, (MouseEvent e) -> {
    button2.setEffect(null);
});
```

Evento de teclado registrado mediante método de conveniencia
KeyEvent.KEY_PRESSED

```
scene.setOnKeyPressed((KeyEvent event) -> {
    if (event.getCode() == KeyCode.ESCAPE) {
            stage.close();
    }
});
```

Manejadores mediante referencias a métodos

```
@Override
  public void start(Stage stage) {
...
      button1.setId("B1");
      button2.setId("B2");
      button1.setOnAction((ActionEvent e) -> buttonClicked(e));
      button2.setOnAction(this::buttonClicked);
       …
}
```

Sintaxis alternativa (method reference)

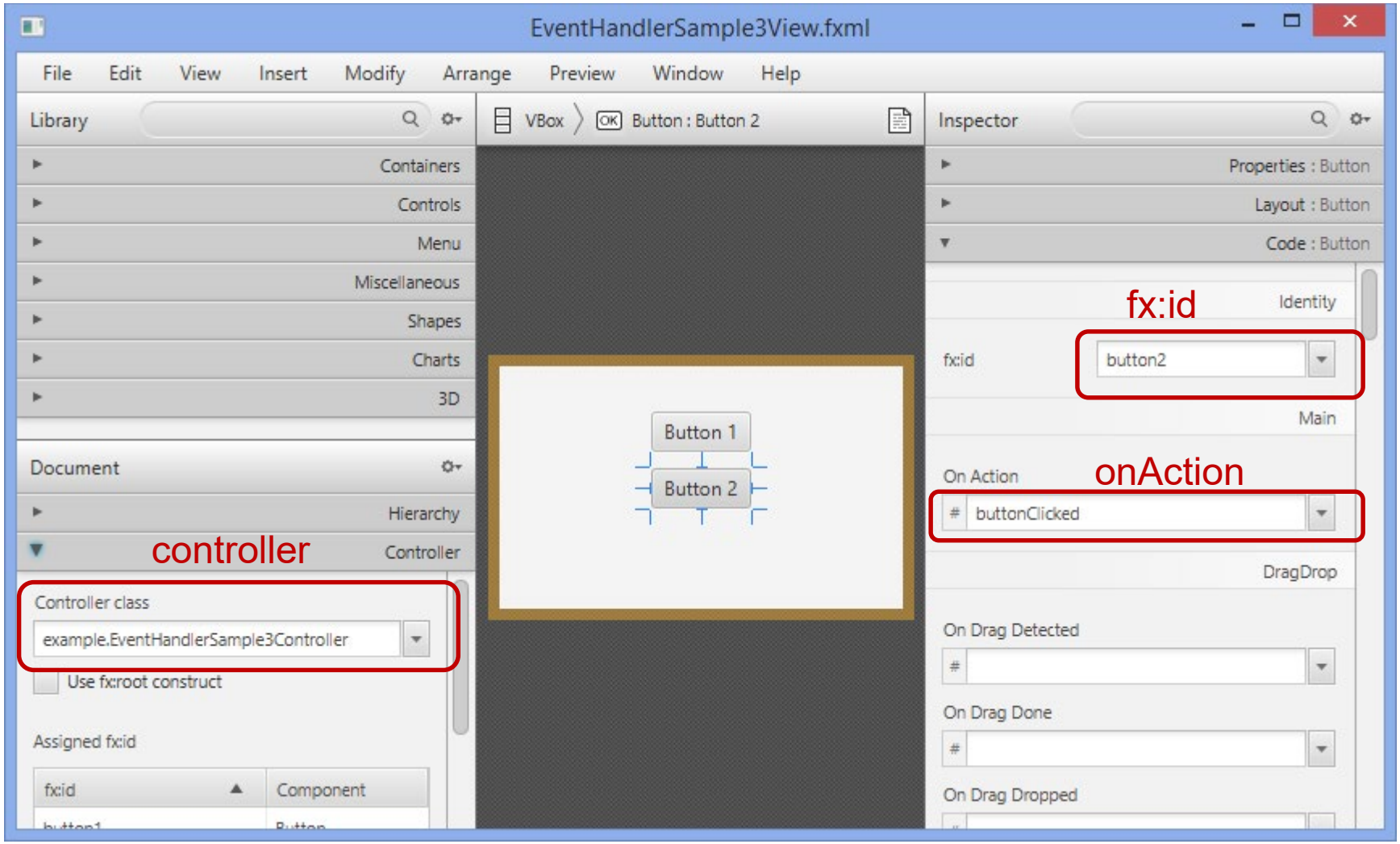
Manejador definido como método independiente

```
void buttonClicked(ActionEvent event) {
  Button button = (Button) event.getSource();
  String id = button.getId();
  label.setText(button.getText() + " (" + id +")");
}
```

Uso de identificadores para distinguir la fuente del evento

```
void buttonClicked(ActionEvent event) {
      String id = ((Node) event.getSource()).getId();
      if (id.equals("B1")) {
         label.setText("Button 1");
      } else {
         label.setText("Button 2");
      } }
```

Eventos en SceneBuilder



Ejemplo 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>
```

Ejemplo FXML + Controlador

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

```
@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);
   }
}
```

Ejercicio

- El objetivo del proyecto es trabajar con eventos de teclado.
  - Crear un proyecto JavaFX nuevo
  - Añadir al grafo de escena un gridpane de 5x5 celdas.
  - Añadir un circulo en el centro del grid.
  - Añadir la gestión de eventos para poder mover el botón mediante las teclas de scroll.

- Trabajo en el laboratorio.

- get{Row,Column}Index (Node child)
- set{Row,Column}Index (Node child)

Ejercicio

- El objetivo del proyecto es trabajar con eventos de teclado.
  - Crear un proyecto JavaFX nuevo
  - Añadir al grafo de escena un gridpane de 5x5 celdas.
  - Añadir un circulo en el centro del grid.
  - Añadir la gestión de eventos para poder mover el botón mediante las teclas de scroll.

- Trabajo en el laboratorio.

JavaBeans y Propiedades

- En POO, una **propiedad** es una forma de encapsular información que define una interfaz común para su manejo:
  - Métodos públicos de acceso y modificación denominados *get/set* + NombrePropiedad
- En Java la noción de propiedad no existe como característica propia del lenguaje, pero sí es un elemento de diseño fundamental en la especificación de ***JavaBeans***

```
public class Node {
   private String id;

   public String getId() {
      return id;
   }
   public void setId(String value) {
      id = value;
   }
}
```

Ejemplo de clase Java conforme con el modelo JavaBeans

Propiedades JavaFX

- Una propiedad JavaFX es un tipo de objetos que envuelve o encapsula a otro (patrón “wrapper”), al cual aporta cierta funcionalidad adicional.
- Las clases que tienen *properties* en la API de JavaFX siguen el patrón de JavaBeans con el añadido de un tercer método que devuelve la propiedad (y no su valor)

```
public class Node {
   private StringProperty id = new SimpleStringProperty();
   public String getId() {
      return id.get();
   }
    public void setId(String value) {
      id.set(value);
   }
    public StringProperty idProperty() {
      return id;
   }
}
```

Uso de propiedades en componentes JavaFX

Propiedades JavaFX

- Aunque se pueden crear propiedades envoltorio para cualquier clase, JavaFX define propiedades para todos los tipos primitivos, cadenas y colecciones.

  `StringProperty`

  `IntegerProperty`

  `DoubleProperty`

  `BooleanProperty`

- Para objetos genéricos disponemos de la clase

  `ObjectProperty<T>`

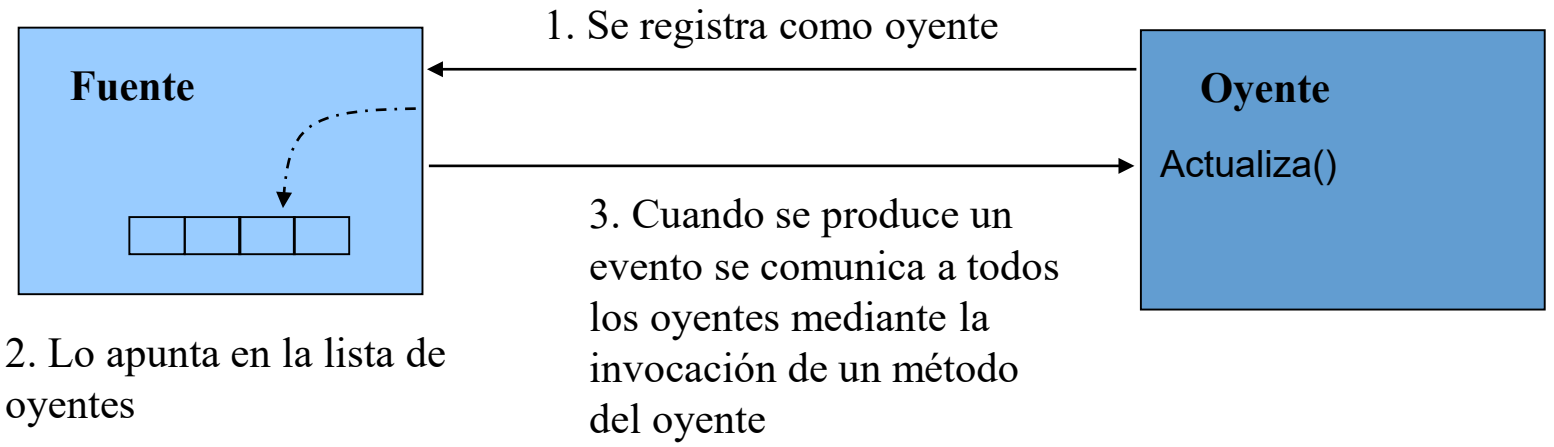
- Para colecciones disponemos de

  `ListProperty`

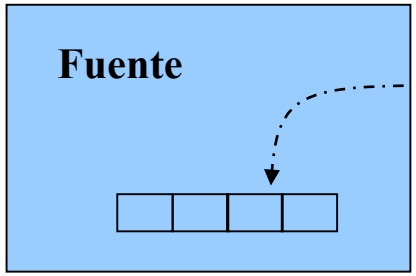
  `MapProperty`

Atendiendo a cambios en las propiedades: Patrón observador

- Dependencia de *uno-a-muchos* entre objetos: cuando un objeto (el sujeto u objeto observable) cambia su estado, los objetos dependientes (observadores) son avisados del cambio



Objetos observables



- Implementa la interfaz ObservableValue<T>
  - T es el tipo de valor que se quiere “observar”
  - Especifica 3 métodos:

`void addListener(ChangeListener<? super T> listener)` añade un nuevo oyente, el cual será notificado de cambios en el valor del objeto observable

`void removeListener(ChangeListener<? super T> listener)` borra de la lista de oyentes al oyente pasado como parámetro.

`T getValue()` Devuelve el valor actual del objeto observable

Oyentes

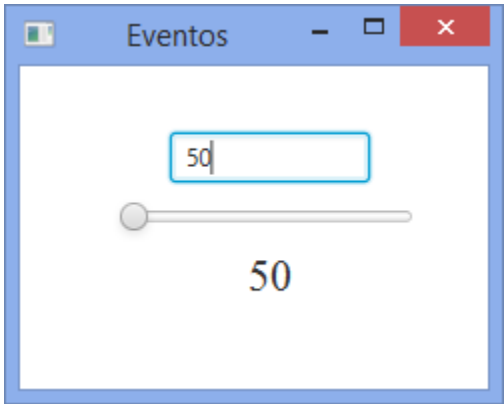
Oyente

- Implementa interfaz ChangeListener<T>, siendo T la clase de valor observado
  - Define un único método:

```
changed(ObservableValue<? extends T> observable, T oldValue, T newValue)
```

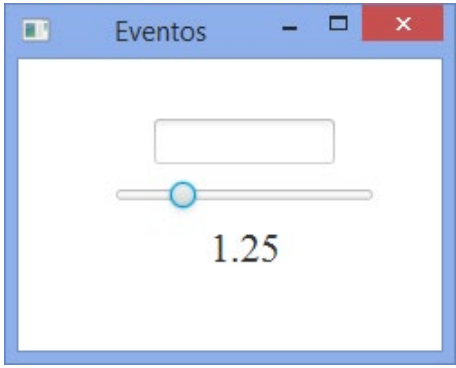
- Muchos controles JavaFX contienen algún atributo de tipo Property<T>, derivado de ObservableValue<T>, y por tanto permiten añadir un ChangeListener<T>
  - Puede añadirse código oyente que será informado cuando el valor de una propiedad cambia.

Ejemplo TextField y Slider




```
Label label = new Label();
label.setFont(Font.font("Times New Roman", 22));
TextField textField = new TextField();
textField.setMaxWidth(100);
Slider slider = new Slider(0, 5, 0);
slider.setBlockIncrement(0.5);
slider.setMaxWidth(150);
void initialize…
  textField.textProperty().addListener(new ChangeListener() {

    @Override
    public void changed(ObservableValue o, Object oldVal, Object newVal) {
      label.setText(newVal + "");
    }
  });

  slider.valueProperty().addListener((observable, oldVal, newVal) ->
     { label.setText(newVal + ""); });
```

Oyente de TextField (clase anónima)

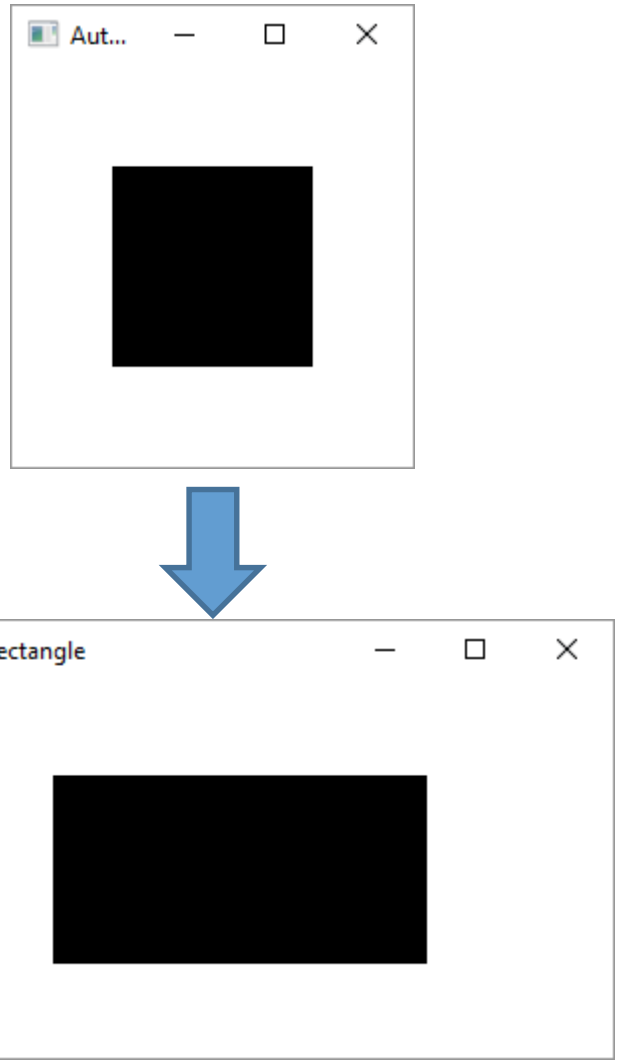


Oyente de Slider (función lambda)

Otro ejemplo con altura y anchura

```
public void start(Stage primaryStage)
   {
      Rectangle r = new Rectangle(100,100);
       StackPane p = new StackPane();
       p.setPrefWidth(200);
      p.setPrefHeight(200);
      p.getChildren().add(r);
      p.widthProperty().addListener(
         (observable, oldvalue, newvalue) ->
              r.setWidth((Double)newvalue/2)
         );
      p.heightProperty().addListener(
         (observable, oldvalue, newvalue) ->
              r.setHeight((Double)newvalue/2)
         );
      Scene scene = new Scene(p);
      primaryStage.setScene(scene);
      primaryStage.setTitle("Auto Rectangle");
      primaryStage.show();
   }
}
```

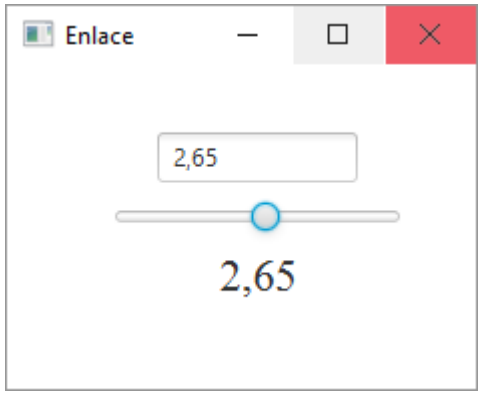
Oyentes



Enlace

- Enlace unidireccional: Si p1 se enlaza unidireccionalmente con p2, p1 tomará siempre el nuevo valor de p2.
  - `p1.bind(p2);`
  - Intentar cambiar el valor de p1 provoca una excepción
- Enlace bidireccional: los cambios en una propiedad se replican en la otra (equivale a enlazar unilateralmente p1 con p2 y p2 con p1)
  - `p1.bindBidirectional(p2);`
- Los enlaces se crean con los métodos *bind/bindBidirectional* y se deshacen con *unbind/unbindBidirectional*.

Ejemplo *TextField* y *Slider*




```
Label label = new Label();
label.setFont(Font.font("Times New Roman", 22));
TextField textField = new TextField();
textField.setMaxWidth(100);
Slider slider = new Slider(0, 5, 0);
slider.setBlockIncrement(0.5);
slider.setMaxWidth(150);
label.textProperty().bind(textField.textProperty());
textField.textProperty().bindBidirectional(slider.valueProperty(), new Numbe
// label.textProperty().bind(Bindings.format("%.2f", slider.valueProperty()));
```

Enlace unidireccional

Enlace bidireccional

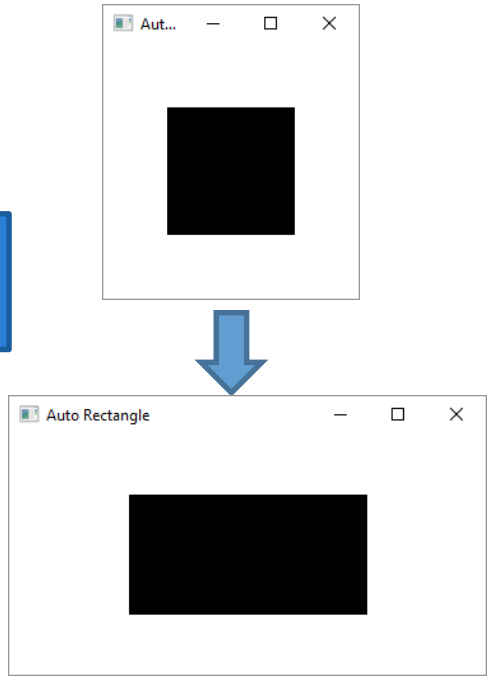
Conversor de número a cadena

Bindings

- Clase auxiliar con muchas funciones de utilidad: ver documentación aquí

```
public void start(Stage primaryStage) {
        Rectangle r = new Rectangle(100, 100);
        StackPane p = new StackPane();
        p.setPrefWidth(200);
        p.setPrefHeight(200);
        p.getChildren().add(r);
        r.widthProperty().bind(
                Bindings.divide(p.widthProperty(), 2));
        r.heightProperty().bind(
                Bindings.divide(p.heightProperty(), 2));
        Scene scene = new Scene(p);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Auto Rectangle");
        primaryStage.show();
    }
```

Ejemplo Bindings.divide(…)



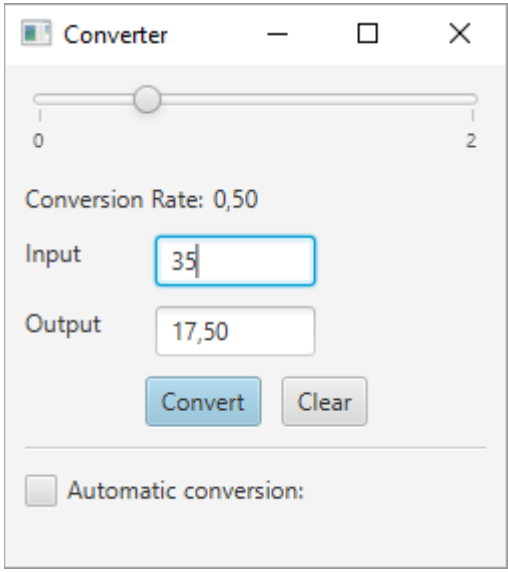
Actividad

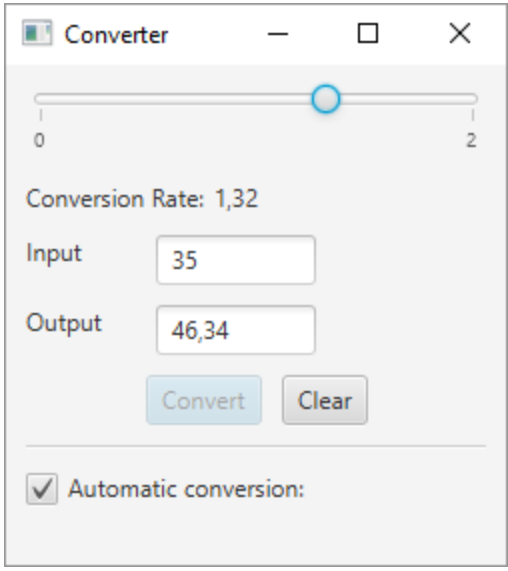
Completar la aplicación de la imagen, con los siguientes requisitos:

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.

Actividad

- App que multiplica un valor de entrada por un cierto ratio de conversión para obtener un valor de salida.
- El ratio de conversión se introduce mediante un *slider*, cuyo valor se muestra debajo en un *label* (usar **enlace**)
- En modo normal hay que pulsar el botón *Convert* para obtener el resultado. El botón *Clear* borra los valores de entrada y salida
- En modo automático se recalcula el resultado tanto si cambia el valor de entrada como si cambia el ratio de conversión (**oyente**)





Referencias

- Tutorial Oracle: Handling JavaFX Events http://docs.oracle.com/javafx/2/events/jfxpub-events.htm
- API JavaFX 8: https://docs.oracle.com/javase/8/javafx/api/
- 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*