



# P2. INTERACCIÓN CON EL USUARIO (PARTE II)

---

Interfaces Persona Computador

Depto. Sistemas Informáticos y Computación

UPV

# Índice

- Propiedades JavaFX
- Listener
- JavaBeans y propiedades
- Bind
- Fluent API y Bindings

# Propiedades JavaFX – Property<T>

Una propiedad JavaFX es una clase wrapper que (1) **envuelve** un campo de un objeto (atributo) y que le añade funcionalidad extra: es (2) **observable**, y permite (3) **enlaces** entre propiedades similares.

- (1) **Envuelve**: en lugar de un atributo de tipo entero tendremos un objeto del tipo Property que envuelve a un entero

int

miEntero;



```
IntegerProperty envoltorioMiEntero = new SimpleIntegerProperty();
```

- Property<T> es una interface, hay una implementación para cada tipo de datos básico de java

# Propiedades JavaFX

- Implementación de **Property** sobre los tipos primitivos.

```
StringProperty st = new SimpleStringProperty();//sobre String  
IntegerProperty in = new SimpleIntegerProperty();//sobre int  
DoubleProperty dob = new SimpleDoubleProperty(); //sobre double  
BooleanProperty bo = new SimpleBooleanProperty();//sobre boolean
```

- Para objetos genéricos disponemos de la clase

```
ObjectProperty<T> obp= new SimpleObjectProperty <>();//complejos
```

- Para colecciones disponemos de

```
ListProperty
```

```
MapProperty
```

# Propiedades JavaFX – Property<T>

`int miEntero;`  `IntegerProperty envoltorioMiEntero = new SimpleIntegerProperty();`

- Para acceder al valor envuelto por la propiedad, tenemos dos métodos:

`get()` ; // devuelve el valor, como tipo básico, es decir con int en el ejemplo

`getValue();` // devuelve el valor como objeto, como Number




- Y para modificar el valor:

`set();` // como tipo básico, es decir con int en el ejemplo

`setValue();` // como Number

- En el caso de `StringProperty` y `ObjectProperty` siempre se utilizan referencias por ser objetos

# Propiedades JavaFX

<code>int miEntero;</code>		<code>IntegerProperty envoltorioMiEntero = new SimpleIntegerProperty();</code>
		<code>// para modificar el valor que esta dentro de la propiedad setValue()</code>
<code>miEntero= 4+5;</code>		<code>EnvoltorioMiEntero.setValue(4+ 5);</code>
		<code>// para obtener el valor que esta dentro de la propiedad getValue()</code>
<code>println(miEntero);</code>		<code>println(envoltorioMiEntero.getValue());</code>

# Propiedades JavaFX: Listener

Una propiedad JavaFX es una clase wrapper que (1) envuelve un campo de un objeto (atributo) y que le añade funcionalidad extra: es (2) observable, y permite (3) enlaces entre propiedades similares.

- **(2) Observable:** Las propiedades cuentan con el método `addListener()` que podemos usar para registrar un método (similar al manejadores de evento), que será notificado (ejecutado) cuando el valor de la propiedad cambie

```
IntegerProperty miEntero= new SimpleIntegerProperty();  
  
miEntero.addListener(this::listenerMiEntero);  
}  
private void listenerMiEntero(ObservableValue<? extends Number> obs, Number oldValue, Number newValue){  
    System.out.println("miEntero ha cambiado. Valor antiguo: " + oldValue + " valor nuevo: "+newValue);  
}
```

# Manejador del cambio, Listener:

- Un manejador de cambio es un método con la interfaz `ChangeListener<T>` siendo `T` el tipo de la propiedad, la cabecera del método es:

```
void change(ObservableValue<? extends T> ob, T oldValue, T  
newValue)
```

Es un método con tres parámetros:

- el primero es la **propiedad** (similar al source en el evento),
- el segundo es el **valor antes de cambiar**, y
- el tercero es el **valor actual**

```
private void listenerMiEntero(ObservableValue<? extends Number> obs, Number oldValue, Number newValue){  
    System.out.println("miEntero ha cambiado. Valor antiguo: " + oldValue + " valor nuevo: "+newValue);  
}
```



# Registro de un listener siempre por código

- Para registrar y recibir notificaciones para tratar los cambios:
  - `addListener( )`

```
miEntero.addListener(this::listenerMiEntero);
```

- Cuando no queremos recibir más notificaciones:
  - `removeListener()`

```
miEntero.removeListener(this::listenerMiEntero);
```

# Registro de un listener con lambda

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  
}
```

```
miEntero.addListener((ob, oldV, newV) -> {
    System.out.println("miEntero ha cambiado. Valor antiguo: " + oldV + " valor nuevo: " + newV);
});
```

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

# JavaBeans y Propiedades

- JavaBeans es un modelo de componentes en el que los atributos de una clase se definen como privados y se sigue un convenio para definir los métodos públicos para acceder a estos atributos, o **propiedades**.

- Modificar: `set + NombrePropiedad ();`
- Recuperar: `get + NombrePropiedad();`
- La propiedad: `nombrePropiedad + Property();`  
( para poder añadir listener )

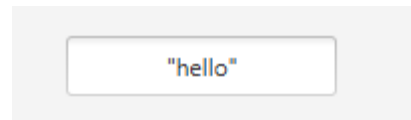
```
public class Node {  
    private StringProperty id = new SimpleStringProperty();  
  
    public String getId() {  
        return id.get();  
    }  
    public void setId(String value) {  
        id.set(value);  
    }  
    public StringProperty idProperty() {  
        return id;  
    }  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

Ejemplo de clase Java conforme  
con el modelo JavaBeans

# JavaBeans y Propiedades

- Los controles de JavaFX usan propiedades para todos sus campos.
- El control TextField tiene varias propiedades, una de ellas es text, para esta propiedad tenemos los siguientes métodos:

- `text.textProperty();`
- `text.setValue();`
- `text.getValueSafe();`
- `text.setText();`
- `text.getText();`



`textProperty()`



# 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);
```

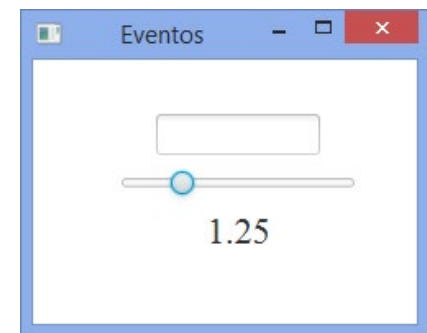
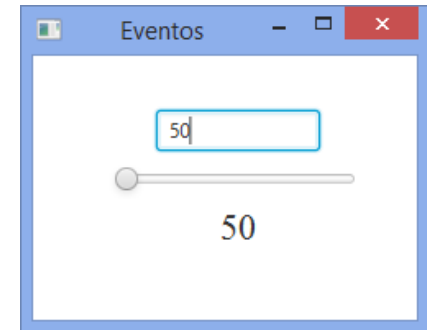
Oyente de TextField  
(función lambda)

void initialize...

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

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

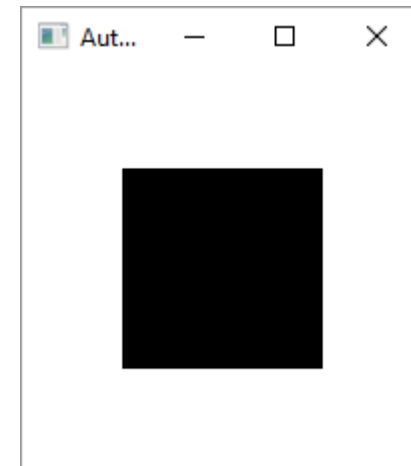
Oyente de Slider  
(función lambda)



# 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, oldval, newval) ->
            r.setWidth((Double)newval/2)
    );
    p.heightProperty().addListener(
        (observable, oldval, newval) ->
            r.setHeight((Double)newval/2)
    );
    Scene scene = new Scene(p);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Auto Rectangle");
    primaryStage.show();
}
```

Oyentes



# Propiedades JavaFX: Bind

Una propiedad JavaFX es una clase wrapper que (1) envuelve un campo de un objeto (atributo) y que le añade funcionalidad extra: es (2) observable, y permite (3) enlaces entre propiedades similares.

- **(3) Enlace**: dos propiedades pueden mantener sus valores sincronizados
  - si la propiedad **propA** esta enlazada con la propiedad **propB** cualquier cambio en el valor de **propB** se reflejará en **propA**, no funciona al inverso, a esto se le llama: ***unidirectional binding***,

```
propA.bind( propB);
```

- si cualquier cambio de **A** se refleje en **B** y un cambio en **B** se refleja en **A** necesitamos un ***bidirectional binding***

```
propA.bindBidirectional( propB );
```

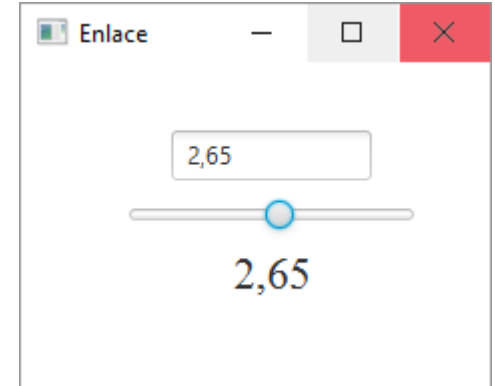
- Para romper la sincronización:

```
propA.unbind();    propA.unbindBidirectional()
```

- Una propiedad enlazada no se puede modificar con `set()`, o `setValue()`

# 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 NumberStringConverter());
```



Enlace unidireccional

Conversor de número a cadena

```
// label.textProperty().bind(Bindings.format("%.2f", slider.valueProperty()));
```

Enlace  
bidireccional



# Fluent API y Bindings

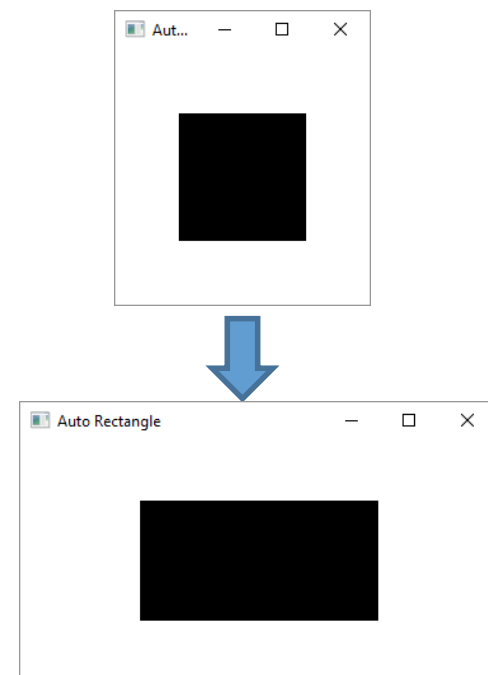
- Fluent API permite hacer operaciones sobre una propiedad mediante métodos como: `multiply()`, `divide()`, `is Equal()`, etc

```
miSlider.valueProperty().divide(2);
```

- Bindings es una clase auxiliar con muchas funciones de utilidad, complementa a fluent API, ver documentación [aquí](#)

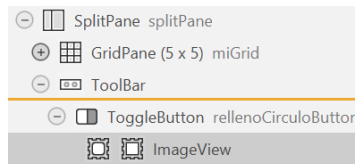
```
miRectangulo.heigthProperty().bind(Bindings.min(miSlider.valueProperty(),2);
```

```
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(p.widthProperty().divide(2));  
    r.heightProperty().bind(Bindings.divide(p.heightProperty(), 2));  
    Scene scene = new Scene(p);  
    primaryStage.setScene(scene);  
    primaryStage.setTitle("Auto Rectangle");  
    primaryStage.show();  
}
```



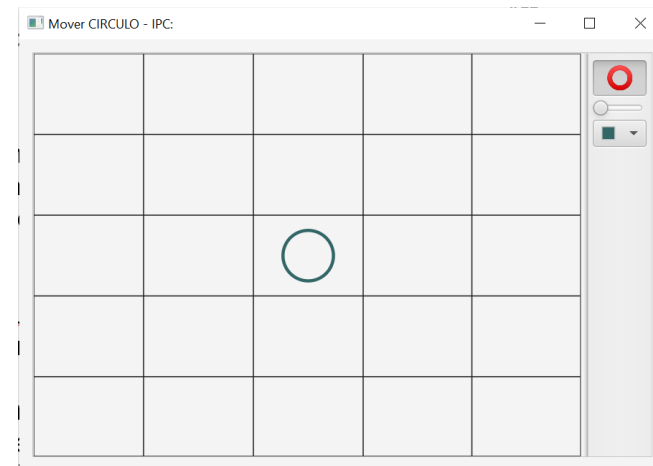
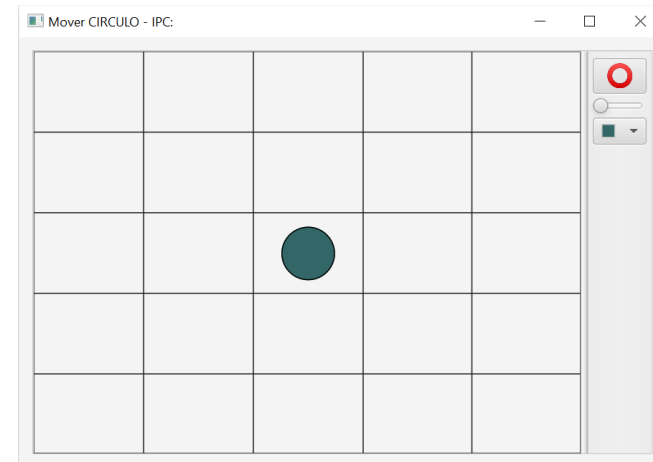
# Ejercicio: (AMPLIACIÓN)

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.
  - Si es un ToolBar modificar orientation= VERTICAL
3. Añadir al ToolBar un ToggleButton con el que gestionar que el círculo este relleno o no. Tamaño 50x25
  - Añade un ImageView dentro del ToggleButton, y en la propiedad Image del ImageView indica el fichero png que se encuentra en el package resources



- Añade como consideres oportuno el siguiente comportamiento sobre el ToggleButton

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

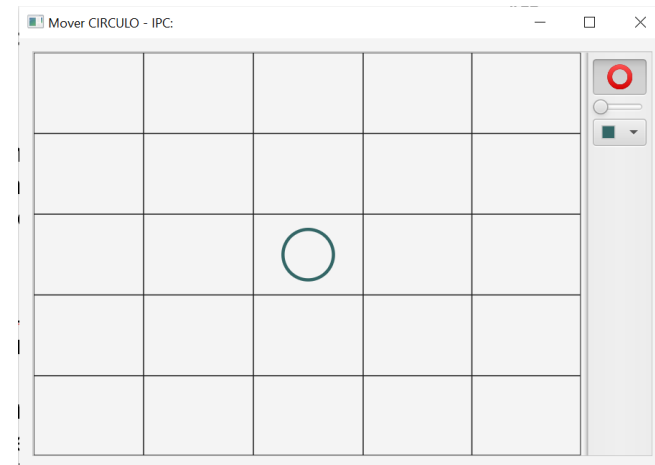
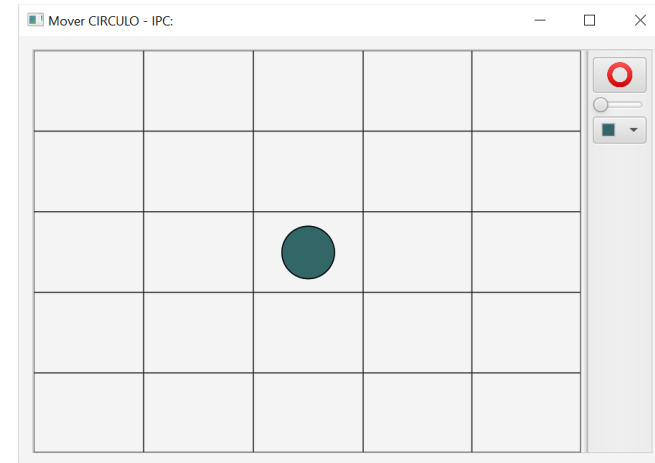


## Ejercicio: (AMPLIACIÓN)

4. Añadir al ToolBar un slider con el que gestionar el tamaño del círculo. Tamaño 50x25
  - Utiliza un enlace
5. Añadir al ToolBar un ColorPicker con el que cambiar el color del círculo. Tamaño 50x25
  - Utiliza un listener, ten en cuenta que puede que tengas que modificar la funcionalidad del ToggleButton

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

Para que los eventos del teclado lleguen solo al círculo es necesario que todos los nodos salvo el círculo 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/11>
- 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*