Transformaciones

La clase Node : Transformaciones

Algunos de los métodos más importantes de la clase Node son:

- getLayoutX(), getLayoutY(): Para obtener la posición inicial del nodo en la escena.
- setTranslateX(), setTranslateY(): Para mover el nodo a lo largo de los ejes X e Y.
- getTranslateX(),getTranslateY(): Obtiene la posición actual en los ejes X e Y.
- setRotate(): Para rotar el nodo.
- setScaleX(), setScaleY(): Para escalar el nodo en los ejes X y Y.

```
public class EjemploNode extends Application {
    @Override
    public void start(Stage stage) {
        Button boton = new Button("Pulsa");  // button es un Node
        boton.setTranslateX(50);  // mover 50 píxeles a la derecha
        boton.setTranslateY(50);  // mover 50 píxeles hacia abajo
        boton.setRotate(45);      // rotar 45 grados
        boton.setScaleX(1.5);     // escalar 1.5 veces en el eje X

        Scene scene = new Scene(boton, 500, 500);
        stage.setScene(scene);
        stage.show();
    }
}
```

Para lograr que el círculo se mueva cada vez que pasamos el ratón por encima, deberemos coger las coordenadas actuales del círculo en cada momento.

```
public class PrimaryController {

    @FXML
    private Circle circuloAzul;


    @FXML
    public void moveCircle(){
        //3- Modifica la posición cada vez
        var coordX = circuloAzul.getTranslateX();

        circuloAzul.setTranslateX(coordX+50);

    }
```

Translate transition

Para implentar este caso podemos utilizar la clase **TranslateTransition**. Algunos métodos interesantes que aporta esta clase son:

- play()
- pause()
- stop()
- onFinished()
- setDuration()
- setDelay()
- setCycleCount()

```
// TranslateTransition
TranslateTransition tt = new TranslateTransition();
```

```
 */
public void initialize() {
    // Auto Reverse
    tt.setAutoReverse(bln:true);
    //Duración transición
    tt.setDuration(drtn: Duration.millis(d: 5000));
    // Valores iniciales
    tt.setByX(d: 500);
    tt.setByY(d: 350);
    //Cycle count
    tt.setCycleCount(i: 500);
    // Nodo
    tt.setNode(node: rectangulo);
    //Ejecutar
    tt.play();
}
```

```
@FXML
private void direccionAleatoria() {

    // Detiene tt anterior
    tt.stop();

    // Cambia texto
    rectangulo.setText("Púlsame: " + contador);

    // Cambia duración
    tt.setDuration(drtn: Duration.millis((int) Math.floor(Math.random() * (1200 - 1000 + 1) + 1000)));

    // Ruta tt aleatoria
    int randomx = (int) Math.floor(Math.random() * ((fondo.getWidth()) - 1 + 1) + 1);
    int randomy = (int) Math.floor(Math.random() * ((fondo.getHeight()) - 1 + 1) + 1);
    tt.setToX(d: randomx);
    tt.setToY(d: randomy);

    // Color aleatorio
    int rgb1 = (int) Math.floor(Math.random() * 255 - 0 + 1 + 0);
    int rgb2 = (int) Math.floor(Math.random() * 255 - 0 + 1 + 0);
    int rgb3 = (int) Math.floor(Math.random() * 255 - 0 + 1 + 0);
    int rgb4 = (int) Math.floor(Math.random() * 255 - 0 + 1 + 0);
    color = "-fx-background-color:rgba(" + rgb1 + "," + rgb2 + "," + rgb3 + "," + rgb4 + ")";
    rectangulo.setStyle(string: color);

    //Ejecutar tt con nuevos parámetros
    tt.play();

    System.out.println(x: "Click");
}
```

En apuntes:

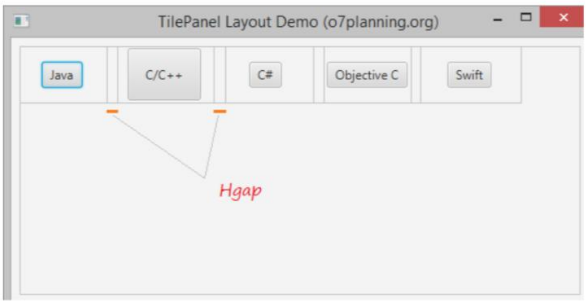
```
TranslateTransition buttonTransition;

@Override
public void initialize(URL url, ResourceBundle rb) {
    buttonTransition = new TranslateTransition(drtn: Duration.millis(d: 11000), node: yellowButton);
    buttonTransition.setByX(d: 600);
    buttonTransition.setByY(d: 600);
    buttonTransition.play();
}

@FXML
public void stopTransition(){ //Función asociada al evento click del botón
   buttonTransition.stop();
}
```

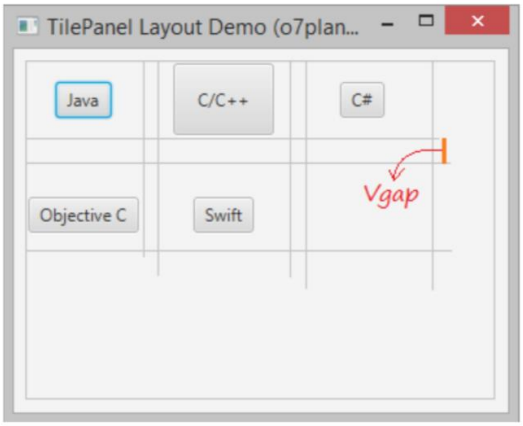
Layouts

TilePane

Layouts: TilePane

Utiliza las propiedades Hgap y VGap para determinar la distancia entre elementos. Cuando éstos tienen distinto tamaño no resulta útil. Por otro lado no aprovecha el borde del layout.

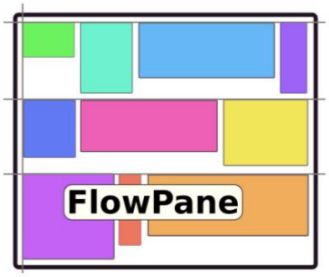


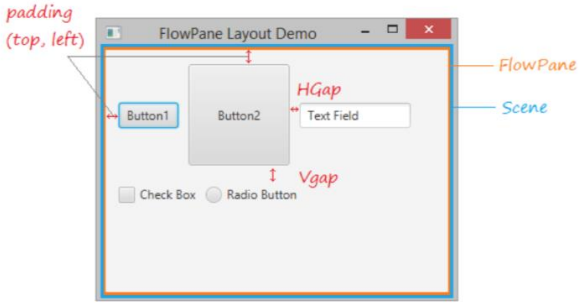


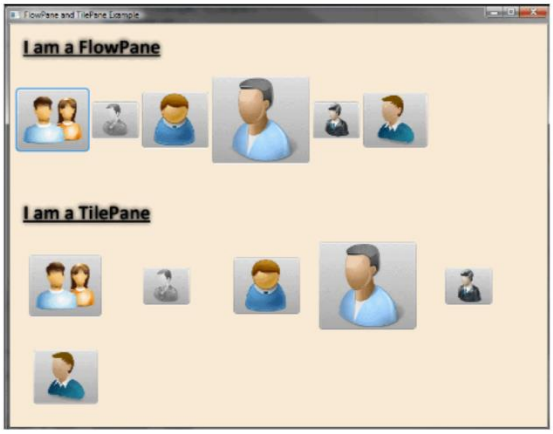
FlowPane

Layouts: FlowPane

- FlowPane: Coloca los elementos en una fila o en una columna y los mueve a la siguiente fila o columna cuando se alcanza el borde del layout. Útil cuando deseas que los nodos tengan tamaños variables y se organicen de manera dinámica en filas y columnas.

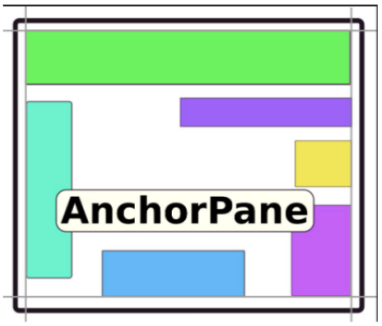




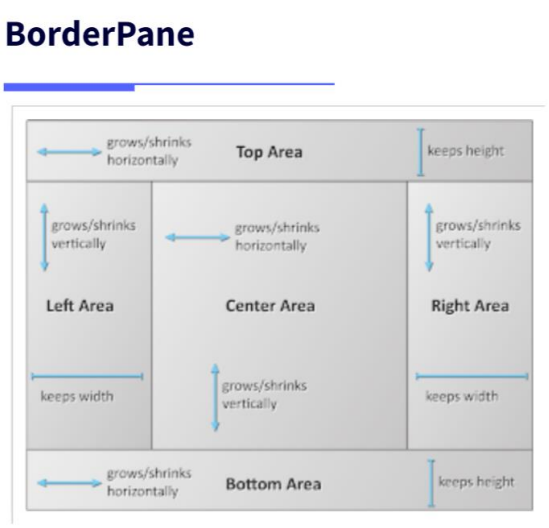


AnchorPane

- AnchorPane: Permite "anclar" los nodos a los bordes del layout utilizando las propiedades AnchorPane.topAnchor, AnchorPane.bottomAnchor, AnchorPane.leftAnchor, y AnchorPane.rightAnchor. Esto puede ser útil para crear diseños más flexibles que se ajustan bien al cambio de tamaño de la ventana.



BorderPane

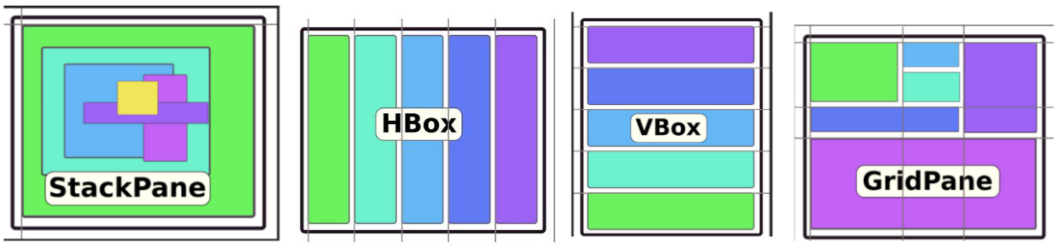


Top y bottom crecen en horizontal

Left y right crecen en vertical

Dentro de cada región otros contenedores, como HBOX, VBOX, GridPane, etc.

Otros



Bindings

Bindea tres textfields con label

```
@FXML
private TextField tf1;
@FXML
private TextField tf2;
@FXML
private TextField tf3;
@FXML
private Label label1;

 public void initialize() {
    label1.textProperty().bind(ov: Bindings.concat(os: tf1.textProperty(),os: " ",os: tf2.textProperty(),os: " ",os: tf3.textProperty()));
 }
```

Bindea botón con checkbox (visibilidad)

```
public void initialize(){
    // HACE VISIBLE AL BOTÓN AL MARCHAR CHECKBOX
    boton.visibleProperty().bind(ov: cb.selectedProperty());


    //este seria para hacerlo que funcione al reves
    // boton.visibleProperty().bind(Bindings.when(cb.selectedProperty()).then(false).otherwise(true));


    //otra forma de hacerlo definiendo una condicion y aplicandola al property

   // ObservableBooleanValue  cond = new SimpleBooleanProperty(false);
    //boton.visibleProperty().bind(cb.selectedProperty().isEqualTo(cond));
```

Visibilidad tf+label

```
// PARA QUE SE VEA EL SEGUNDO LABEL Y TEXTFIELD AL ESCRIBIR EL PRIMERO
// podemos añadir cosas a los property en caso de que no sean directo(bool - bool) , como en este caso
//con el isEmpty hacemos bool la propiedad de text
labelAp.visibleProperty().bind(ov: tf.textProperty().isNotEmpty());
tf2.visibleProperty().bind(ov: tf.textProperty().isNotEmpty());
```

ObservabeList con TableView

```
//ObservableList
static ObservableList<Jugador> valoresLista = FXCollections.observableArrayList();
//Property
private Property<ObservableList<Jugador>> jugadorListProperty = new SimpleObjectProperty<>(b: valoresLista);
```

(en initialize)

```
//Binding: valoresLista - formulario (TableView)
formulario.itemsProperty().bind(ov: jugadorListProperty);
```

Binding y properties

En JavaFX, el binding (enlace) y las propiedades permiten que las variables y los elementos de la UI estén sincronizados.

Una **propiedad** en JavaFX no es más que una clase especial que puede contener un valor, como un int, un String, un objeto, etc. Las propiedades son objetos que encapsulan un valor y permiten ser observados, lo que significa que puedes ser notificado cuando su valor cambia.

```
//podemos declarar propiedades para encapsular Strings, enteros, reales y booleanos
StringProperty s1 = new SimpleStringProperty("Hola");
StringProperty s2 = new SimpleStringProperty("Mundo");
IntegerProperty a = new SimpleIntegerProperty(5);
DoubleProperty euros = new SimpleDoubleProperty(1.123);
BooleanProperty condicion = new SimpleBooleanProperty(true);
```

El Binding permite mantener dos propiedades sincronizadas

```
// sincronizamos las propiedades s1 y s2
s1.bind(s2);
```

Unidireccional: Cuando una propiedad cambia, la otra también lo hace, pero no al revés.

```
StringProperty s1 = new SimpleStringProperty("Hola");
StringProperty s2 = new SimpleStringProperty("Mundo");
System.out.println("Antes del binding: " + s2.get());  // imprime "Mundo"
s2.bind(s1); // aquí es donde el binding ocurre
s1.set("Adiós"); // cambia el valor de s1
System.out.println("Después del binding: " + s2.get()); // imprime "Adiós", s2 está vinculado a s1
s2.set("Fin"); // ERROR, tras en vínculo, no se puede cambiar directamente s2
```

Bidireccional: Ambas propiedades se actualizan cuando cualquiera de ellas cambia

```
StringProperty s1 = new SimpleStringProperty("Hola");
StringProperty s2 = new SimpleStringProperty("Mundo");
System.out.println("s1 antes del cambio: " + s1.get()); // imprime "Hola"
s1.bindBidirectional(s2); // binding bidireccional
s2.set("Adiós"); // cambia el valor de s2
System.out.println("s1 después del cambio: " + s1.get()); // imprime "Adiós"
s1.set("Fin");
System.out.println("s1 después del cambio: " + s2.get()); // imprime "Fin"
```

Tipos de Binding

También podemos utilizar la clase estática Bindings, que proporciona métodos de utilidad como :

- add(ObservableNumberValue a, ObservableNumberValue b): Realiza la suma de a y b.
- subtract(ObservableNumberValue a, ObservableNumberValue b): Realiza la resta de a y b.
- multiply(ObservableNumberValue a, ObservableNumberValue b): Realiza la multiplicación de a y b.
- divide(ObservableNumberValue a, ObservableNumberValue b): Realiza la división de a y b.
- when(BooleanExpression condition): Crea un enlace condicional. Puede usarse en conjunción con then y otherwise para definir los dos posibles resultados.

Tipos de Binding

```
IntegerProperty a = new SimpleIntegerProperty(5);
IntegerProperty b = new SimpleIntegerProperty(2);
NumberBinding suma = Bindings.add(a, b);
System.out.println("Suma: " + suma.getValue()); // Suma: 7

BooleanProperty condicion = new SimpleBooleanProperty(true);
StringProperty resultado = new SimpleStringProperty("");
resultado.bind(Bindings.when(condicion).then("Verdadero").otherwise("Falso"));
System.out.println("Resultado: " + resultado.getValue()); // Resultado: Verdadero
```

Ejemplos Binding : Auto resize de la UI

Para sincronizar controles en la UI, será necesario definir en el controller las respectivas propiedades, y hacer un bind con ellas. Ejemplo:

```
package com.aulanosa.autoresize;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.layout.VBox;
import javafx.scene.shape.Rectangle;

public class PrimaryController implements Initializable {

    @FXML
    private Rectangle blueRectangle;

    @FXML
    private VBox pane;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        blueRectangle.widthProperty().bind(ov: pane.widthProperty());
    }
}
```

Ejemplos Binding : Barra de progreso

```
@FXML
private ProgressBar progressBar = new ProgressBar(d: 0);

@Override
public void initialize(URL url, ResourceBundle rb) {

    DoubleProperty progressValue = new SimpleDoubleProperty(d: 0);

    // Vinculamos la propiedad del progreso al contador
    progressBar.progressProperty().bind(ov: progressValue);

    // Aumentar el contador con un hilo (simulando una tarea en segundo plano)
    new Thread(() -> {
        for (int i = 0; i <= 100; i++) {
            progressValue.set(i / 100.0);
            try {
                Thread.sleep(millis: 100);
            } catch (InterruptedException e) {
            }
        }
    }).start();
}
```