



P1. INTRODUCTION TO JAVAFX 8

Interfaces Persona Computador

Depto. Sistemas Informáticos y Computación

UPV

Outline

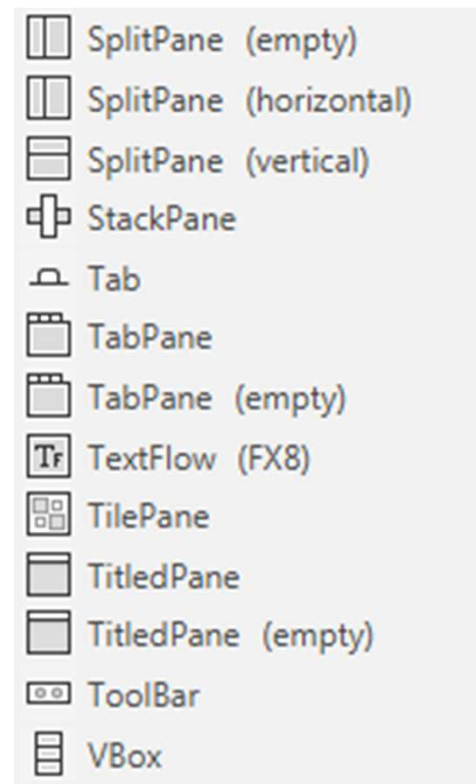
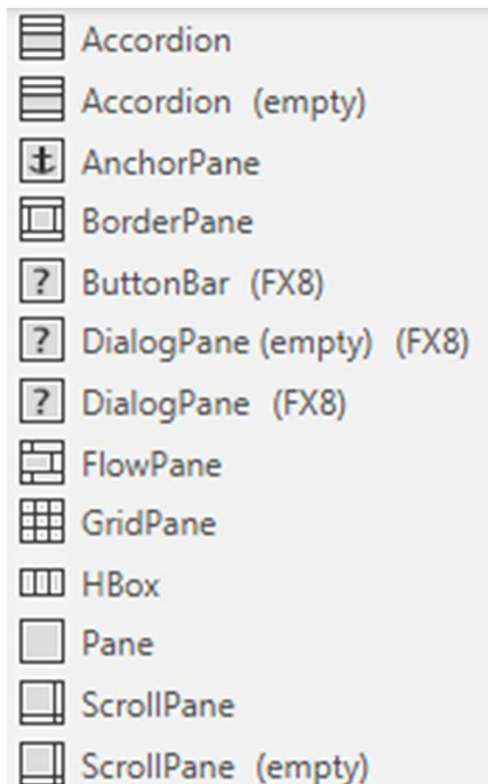
- Introduction
- JavaFX 8 Controls
- Default JavaFX application in NetBeans
- Containers
- Inspecting JavaFX applications
- Exercise

Introduction

- In this session we will study the available controls in JavaFX 8
- We will focus on how to use the containers for organizing the controls of an interface
- We will study how to control the size of the controls
- Finally, we will review how to build the examples in Ensemble using NetBeans and how to inspect running JavaFX applications for studying its scene graph





































JavaFX 8 Controls

- Containers: contain other controls (and containers) and keeps them laid out automatically



JavaFX 8 Controls








- Controls: implement user interaction

 Button	 MenuBar	 Slider (vertical)
 CheckBox	 MenuButton	 Spinner (FX8)
 ChoiceBox	 Pagination	 SplitMenuButton
 ColorPicker	 PasswordField	 TableColumn
 ComboBox	 ProgressBar	 TableView
 DatePicker (FX8)	 ProgressIndicator	 TextArea
 HTMLEditor	 RadioButton	 TextField
 Hyperlink	 ScrollBar (horizontal)	 ToggleButton
 ImageView	 ScrollBar (vertical)	 TreeTableColumn (FX8)
 Label	 Separator (horizontal)	 TreeTableView (FX8)
 ListView	 Separator (vertical)	 TreeView
 MediaView	 Slider (horizontal)	 WebView









JavaFX 8 Controls

- Other controls







Menus

-  CheckMenuItem
-  ContextMenu
-  CustomMenuItem
-  Menu
-  MenuItem
-  RadioMenuItem
-  SeparatorMenuItem

Charts











-  AreaChart
-  BarChart
-  BubbleChart
-  LineChart
-  PieChart
-  ScatterChart
-  StackedAreaChart
-  StackedBarChart

Other

-  Canvas
-  Group
-  Region
-  SubScene (FX8)
-  SwingNode (FX8)
-  Tooltip

2D and 3D graphics

-  AmbientLight (FX8)
-  ParallelCamera (FX8)
-  PerspectiveCamera (FX8)
-  PointLight (FX8)

-  Arc
-  ArcTo
-  Box (FX8)
-  Circle
-  ClosePath
-  CubicCurve
-  CubicCurveTo
-  Cylinder (FX8)
-  Ellipse
-  HLineTo
-  Line
-  LineTo
-  MeshView (FX8)
-  MoveTo
-  Path
-  Polygon
-  Polyline
-  QuadCurve
-  QuadCurveTo
-  Rectangle
-  Sphere (FX8)
-  SVGPath
- Text
- VLineTo

Default JavaFX Application in NetBeans

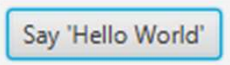

File\New Project\Java FX Application

```
import javafx.application.Application;
/* and other imports */

public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

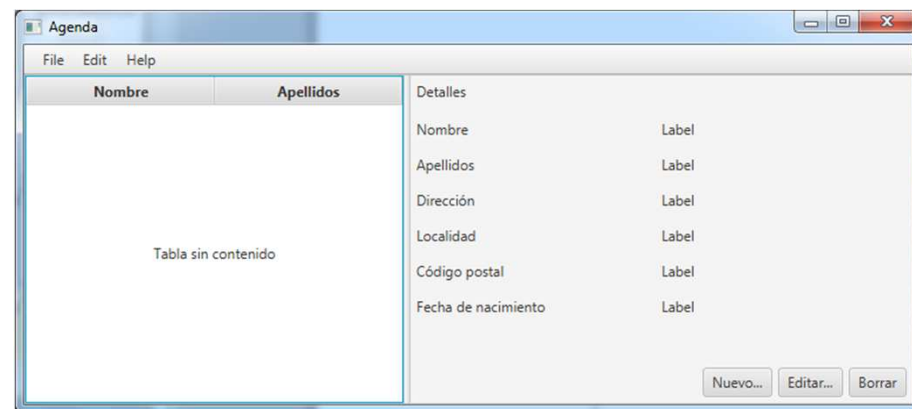
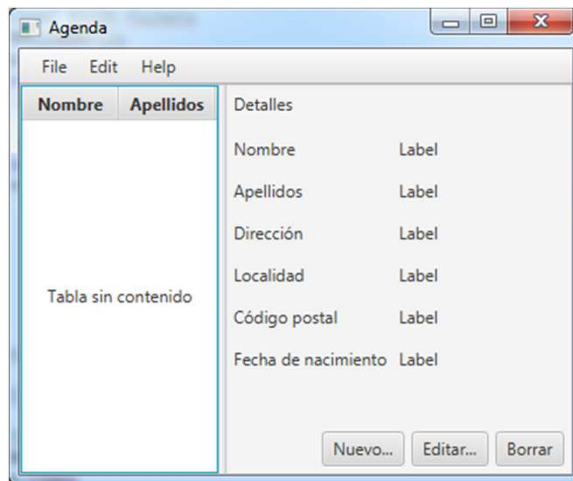
// Creates a button
 // Button label
 // Event handler:
 // the handle method is executed
 // whenever the button is pressed

// Create a container
 // Add the button to the container
 // Create a scene with the container
 // Window title
 // Install the scene in the stage
 // Show the stage

JavaFX Containers

- The developer can set in code the size and position of the widgets in the window, but it is difficult to relocate all the widgets when the window resizes
- It is easier (and mandatory in this course!) to use containers to keep the controls laid out correctly and properly sized for different window sizes



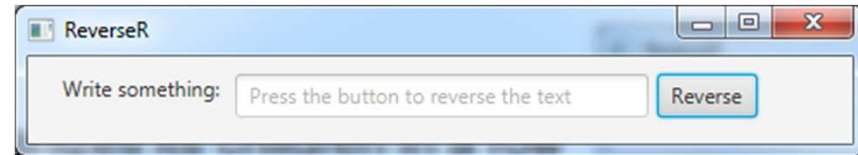
JavaFX Containers

- There are containers that organize their children is rows, columns, stacks, grids, etc.
- The container re-computes automatically the position and size of its children when the window is resized

JavaFX Containers

- HBox

- Organizes its children in a row
- We can set a space around them (padding), and the space between them (spacing)



```
Label lbl = new Label("Write something:");
TextField text = new TextField(); // Text box
text.setPrefColumnCount(20); // Preferred size
text.setPromptText("Press the button to reverse the text"); // Prompt
Button btn = new Button("Reverse");
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent event) {
        String tmp = text.getText();
        text.setText(new StringBuffer(tmp).reverse().toString());
    }
});
HBox root = new HBox(5); // 5 pixels between child.
root.setPadding(new Insets(10)); // 10 pixels around
root.getChildren().addAll(lbl, text, btn);
```

JavaFX Containers

- VBox
 - Similar to HBox, but organizes its children vertically
 - It also defines *padding* and *spacing*



// Same code as before

```
VBox root = new VBox(5);
```

```
root.setPadding(new Insets(10, 10, 10, 20));
```

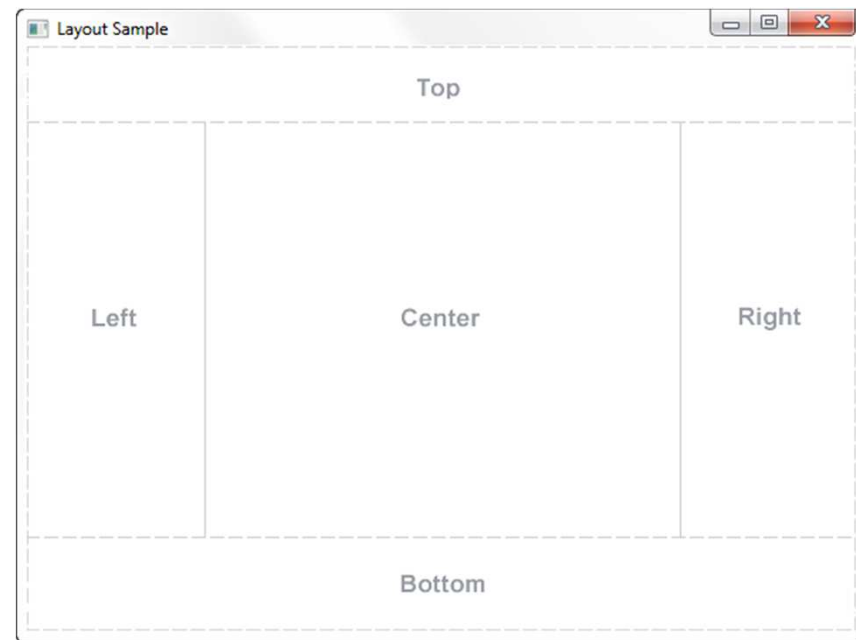
```
root.getChildren().addAll(lbl, text, btn);
```

// 5 pixels between children

// up, right, down, left

JavaFX Containers

- **BorderPane**
- Defines 5 regions:
 - Top, left, center, right and bottom
 - Used for building main windows (with toolbars, status bar, navigation panel on the left and the main work area in the center)
 - If there is enough space, the center grows as needed. If there is not enough space, areas can overlap
 - There can be empty regions

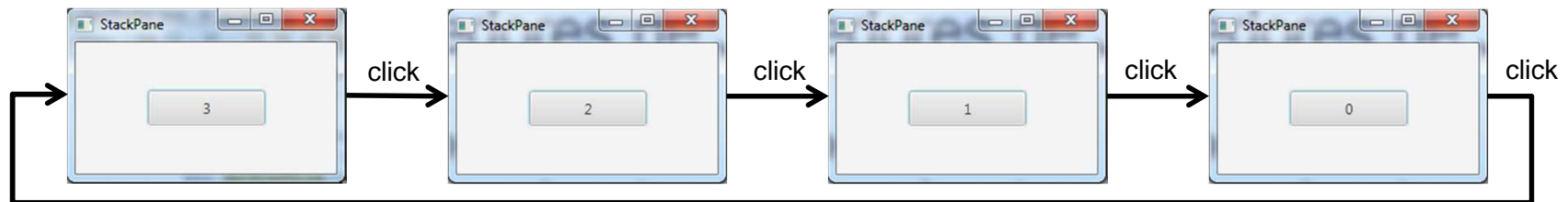


```
BorderPane root = new BorderPane();
root.setTop(new Button("Top"));
HBox group = new HBox();
group.getChildren().addAll(
    new Button("Bottom 1"),
    new Button("Bottom 2"));
root.setBottom(group);
// and setLeft, setCenter and setRight
```

JavaFX Containers

- StackPane
 - Organizes its elements one on top of each other
 - This allows us to overlap text onto other elements (shapes, images, etc.), or showing only an element in a stack

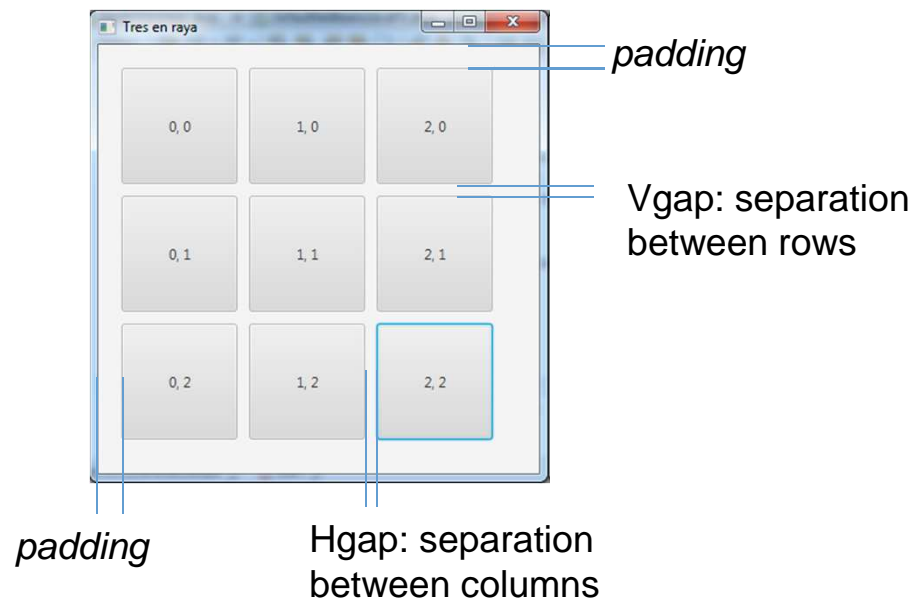
```
StackPane root = new StackPane();
for (int i = 0; i < 4; i++) {
    Button btn = new Button(Integer.toString(i));
    btn.setPrefSize(100, 30);
    btn.setOnAction(new EventHandler<ActionEvent>() {
        @Override public void handle(ActionEvent event) {
            ((Node)event.getSource()).toBack(); // getSource returns who fired the event
                                                // toBack sends the node to the back of the
                                                // container's children list
        }
    });
    root.getChildren().add(btn);
}
```



JavaFX Containers

- GridPane

- Organizes its children in a matrix with rows and columns
- A node can overflow to adjacent cells
- Used to implement forms



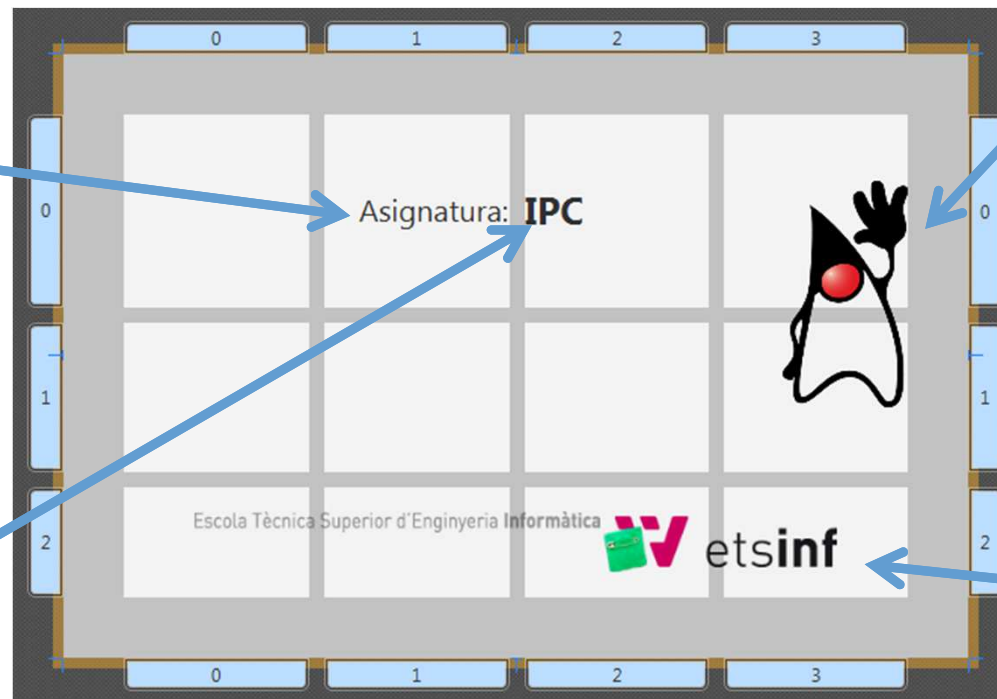
```
GridPane root = new GridPane();
for (int col = 0; col < 3; col++) {
    for (int row = 0; row < 3; row++) {
        Button btn = new Button(
            Integer.toString(col) + ", " + row);
        btn.setPrefSize(100, 100);
        root.getChildren().add(btn);
        GridPane.setConstraints(btn, col, row);
    }
}
root.setVgap(10);
root.setHgap(10);
root.setPadding(new Insets(20));
```

JavaFX Containers

- GridPane: expanding to adjacent cells

Label
 Row Index: 0
 Column Index: 1
 Row Span: 1
 Column Span: 1
 Halignment: RIGHT

Label
 Row Index: 0
 Column Index: 2
 Row Span: 1
 Column Span: 1



ImageView
 Row Index: 0
 Column Index: 3
 Row Span: 2
 Column Span: 1
 Halignment: RIGHT

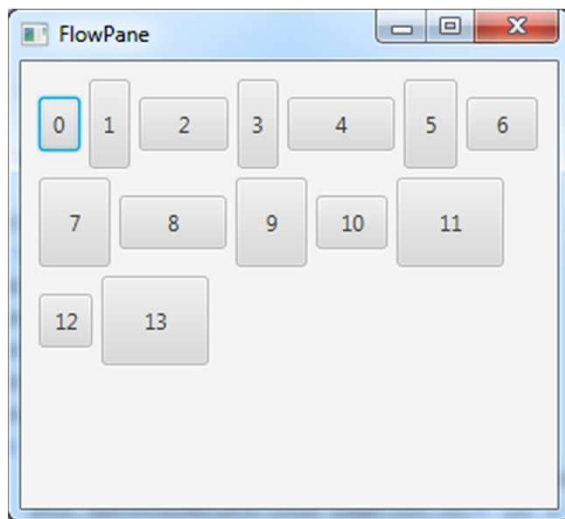
ImageView
 Row Index: 2
 Column Index: 0
 Row Span: 1
 Column Span: 4
 Halignment: CENTER

Method in class GridPane:

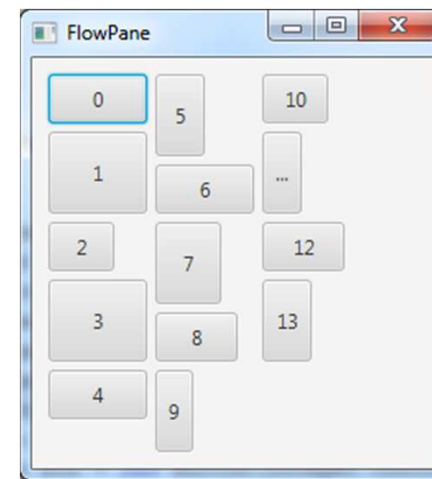
```
public void add(Node child, int columnIndex, int rowIndex, int colspan, int rowspan)
public static void setHalignment(Node child, HPos value) // Hpos.{LEFT,CENTER,RIGHT}
```

JavaFX Containers

- FlowPane
 - Organize its children sequentially, from left to right or top to bottom, adjusting to the container's size



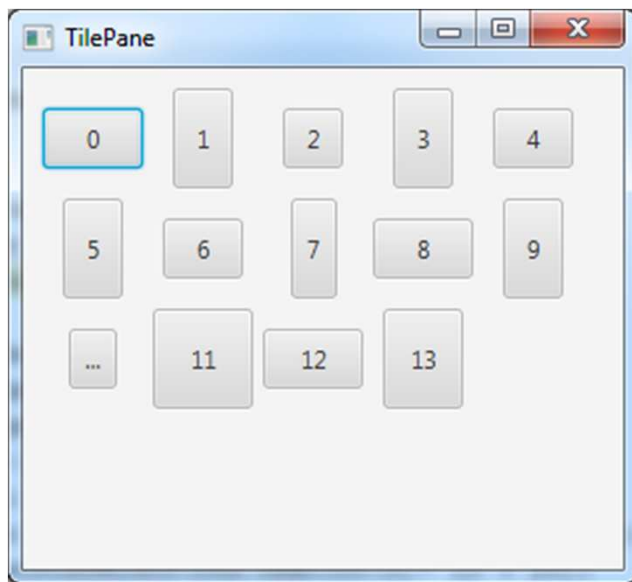
Orientation: HORIZONTAL
(default)



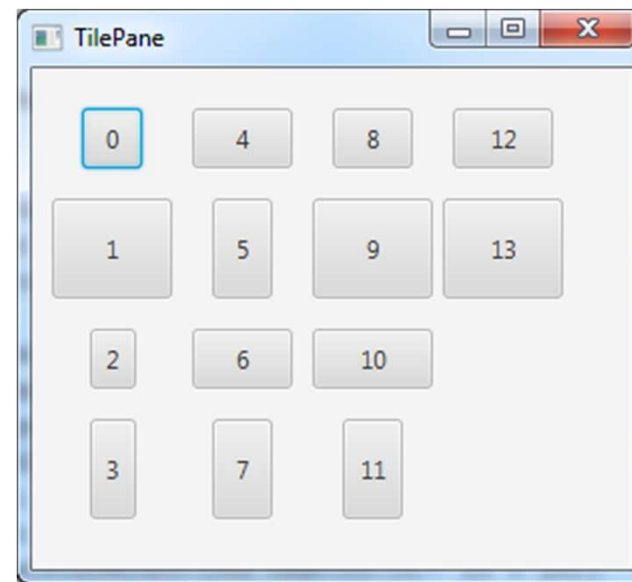
Orientation: VERTICAL

JavaFX Containers

- **TilePane**
 - Similar to FlowPane, but organizes its children sequentially in a matrix where each cell uses the same space, regardless of the child's size



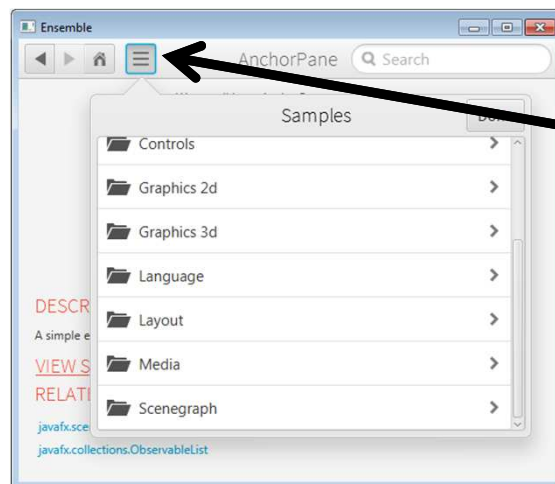
Orientation: HORIZONTAL
(default)



Orientation: VERTICAL

JavaFX Containers

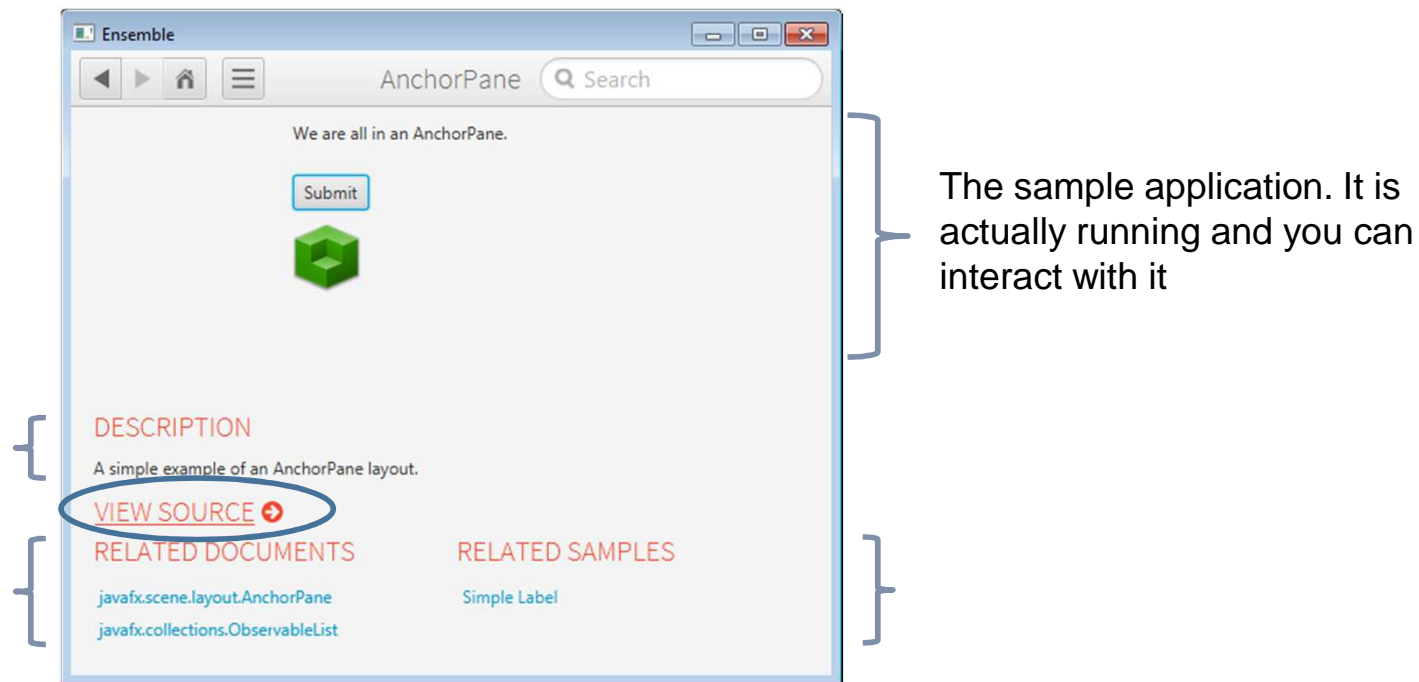
- AnchorPane
 - It allows anchoring their children to a given distance from the borders of the container, or to its center
 - When the window resizes, the nodes keep their relative positions to their anchor points
 - A node can be anchored to more than one anchor point
- Study the examples in Ensemble:



Click on the list icon, select the category “Layout” and open the sample for AnchorPane

JavaFX Containers

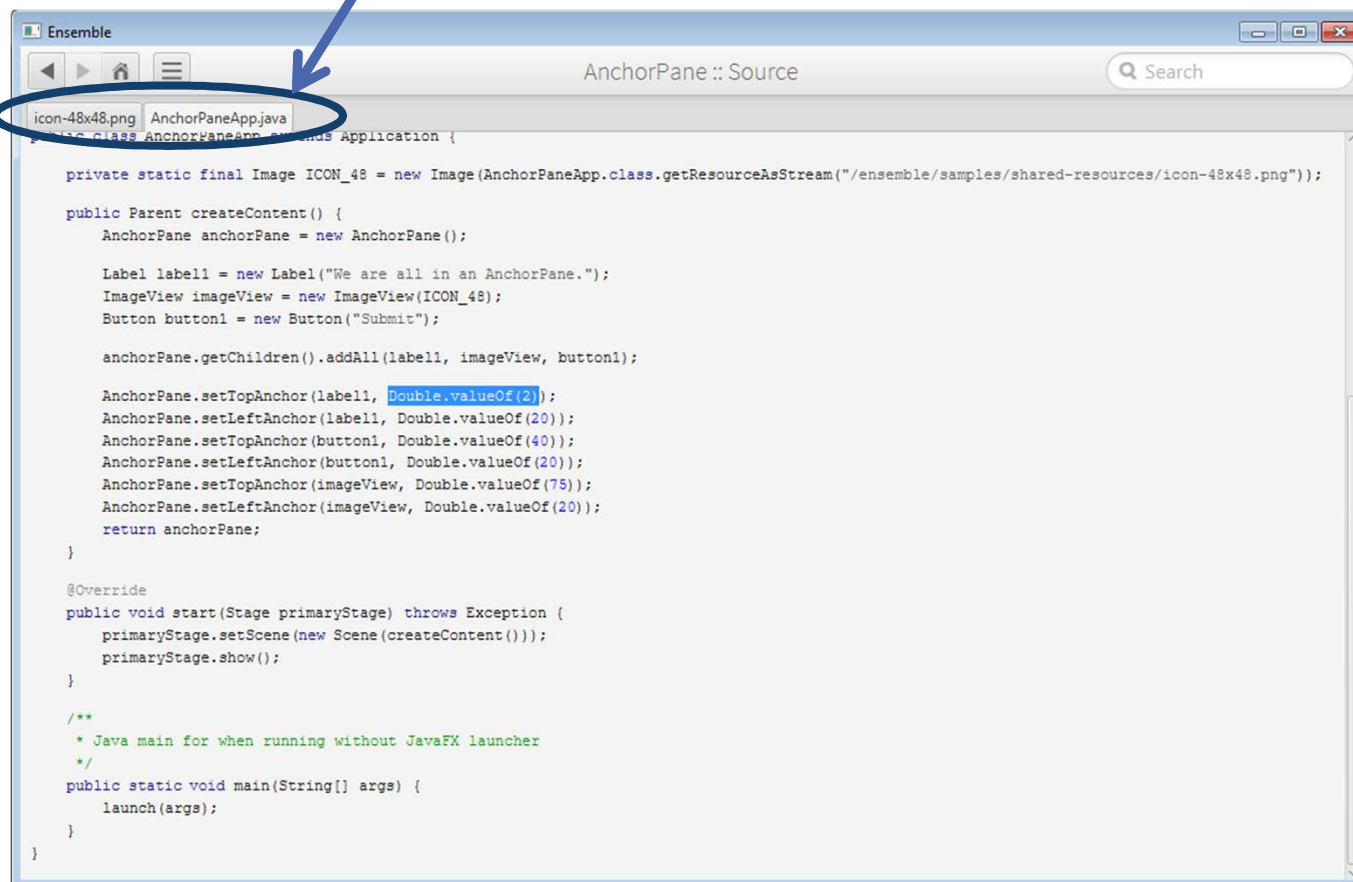
- AnchorPane
 - All the sample pages in Ensemble show:



JavaFX Containers

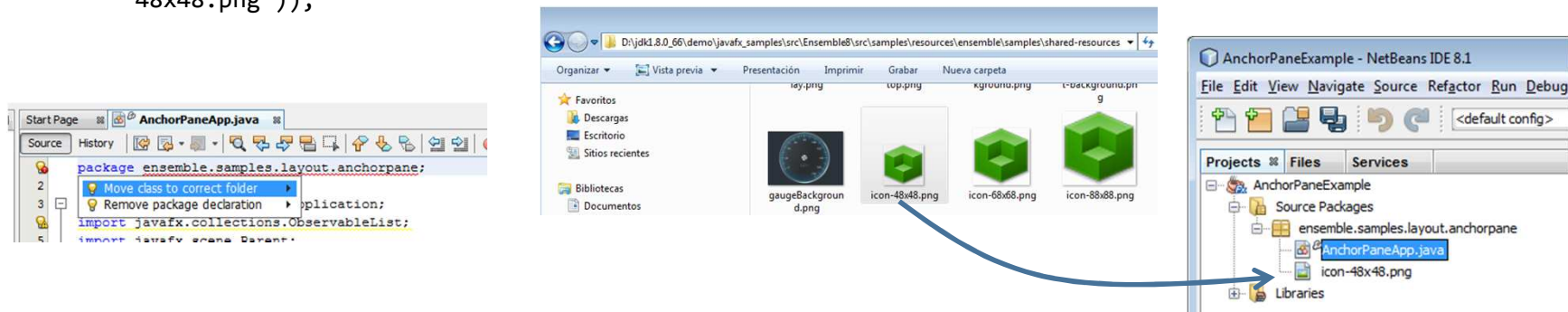
- AnchorPane

A tab per each source file in the sample



JavaFX Containers

- Compiling a sample from Ensemble using NetBeans:
 - Create a new project JavaFX Application
 - Uncheck the option Create Application Class
 - Add to the project a new Java class, using the same class name as the example (AnchorPaneApp)
 - Copy and paste the code in Ensemble to the previous file
 - Move the class to its package
 - Copy other resources, into the package and fix their path in the code:
 - `private static final Image ICON_48 = new Image(AnchorPaneApp.class.getResourceAsStream("/ensemble/samples/shared-resources/icon-48x48.png"));`
 - `private static final Image ICON_48 = new Image(AnchorPaneApp.class.getResourceAsStream("icon-48x48.png"));`




JavaFX Containers

Controlling the size of the nodes

- The advantage of using the JavaFX containers is that they re-compute the position and size of their nodes when the window is resized
- The nodes can be resized according to a number of parameters, presented later
 - There are nodes that do not resize: shapes, text and groups.
- In general, we can specify:
 - a node's size using its preferred size (Pref Width and Pref Height)
 - The position of a node using the alignment properties of the container

JavaFX Containers

Controlling the size of the nodes

- By default, controls compute their preferred size depending on their content
 - The size of a button adapts to its label
- 
- Nodes define minimum and maximum sizes
 - The maximum size of a button, by default, is set to its preferred size (typically we don't want buttons to grow)
 - Other nodes, such as `ListView`, do want to grow to use all the available space.
 - We can set the following sizes
 - Preferred (`setPrefHeight`, `setPrefWidth` o `setPrefSize`)
 - Maximum (`setMaxHeight`, `setMaxWidth`, o `setMaxSize`)
 - Minimum (`setMinHeight`, `setMinWidth`, o `setMinSize`)
 - They all accept values in pixels, or `Double.MAX_VALUE`, `Control.USE_PREF_SIZE` or `Control.USE_COMPUTED_SIZE`

JavaFX Containers

Controlling the size of the nodes

- Homogenizing the node size in a group:

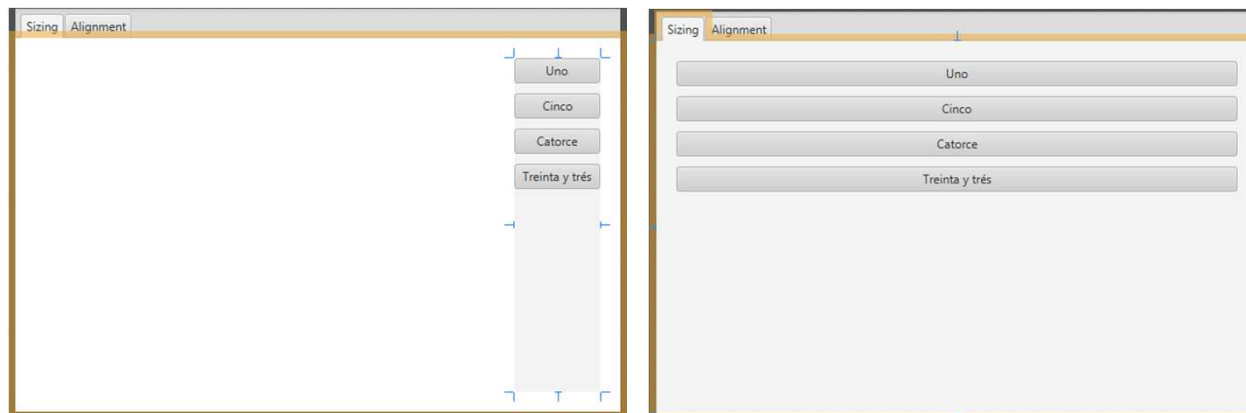
Default maximum

Maximum set to MAX_VALUE



```
VBox root = new VBox();  
Button b1 = new Button("Uno");  
Button b5 = new Button("Cinco");  
[...]  
b1.setMaxWidth(Double.MAX_VALUE);  
b5.setMaxWidth(Double.MAX_VALUE);  
b14.setMaxWidth(Double.MAX_VALUE);  
b33.setMaxWidth(Double.MAX_VALUE);
```

- In a BorderPane, the central area takes all the available space; the other areas only as much space as needed:

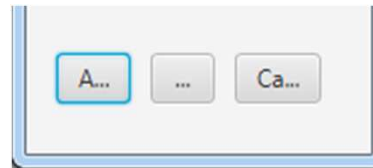
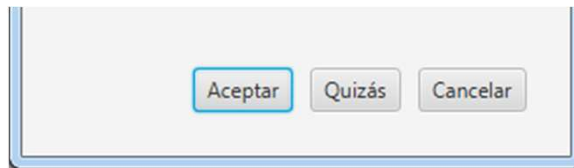


The same VBox as above, in the right area and in the central area of a BorderPane

JavaFX Containers

Controlling the size of the nodes

- For keeping a node from growing:
 - Set the maximum size to `Control.USE_PREF_SIZE`, or to a maximum size in pixels
- For keeping a node from shrinking:
 - Set its minimum size to `Control.USE_PREF_SIZE`

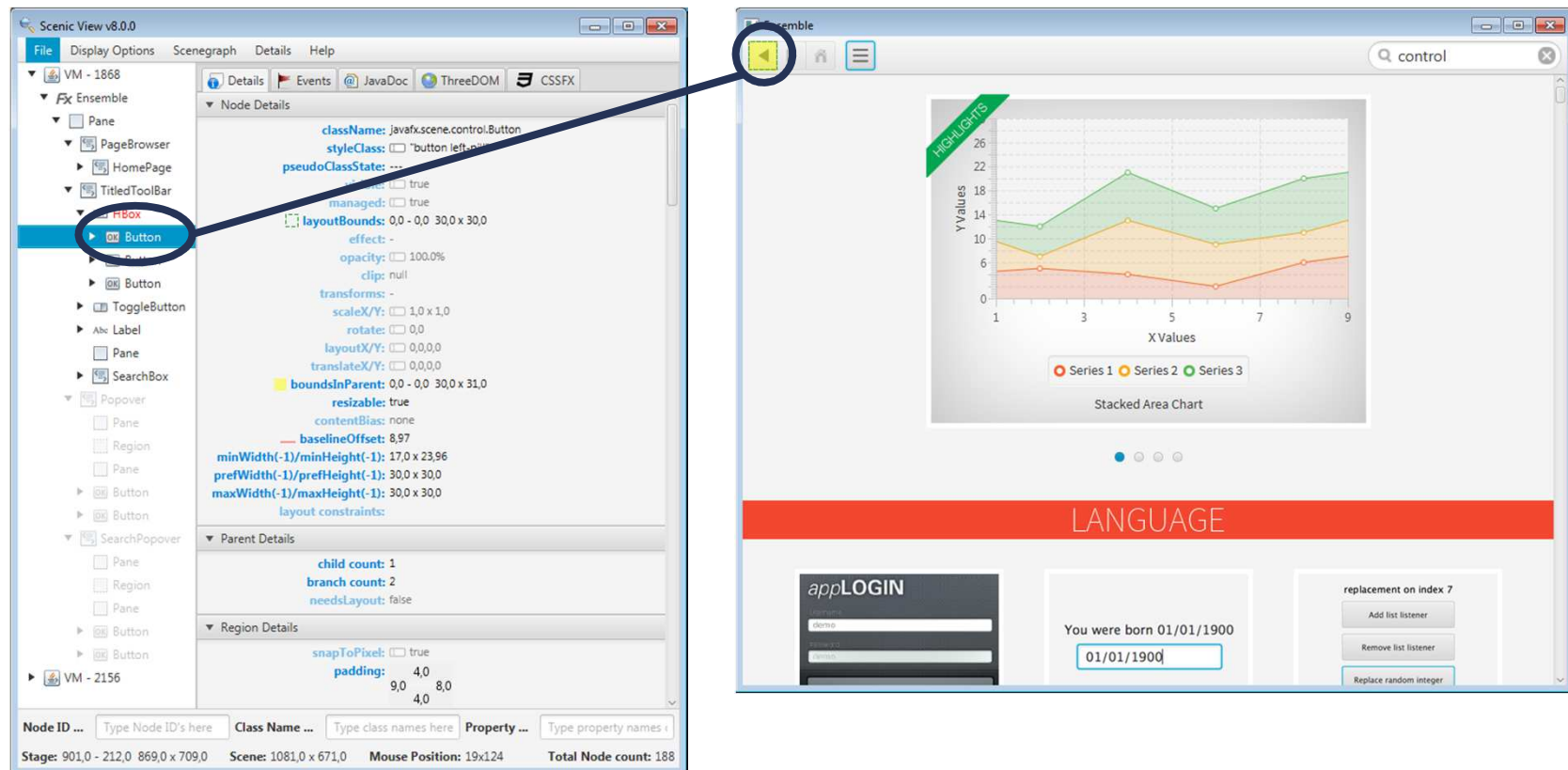


By default, buttons shrink

- For keeping a node from resizing
 - Set the minimum, maximum and preferred sizes to the same value

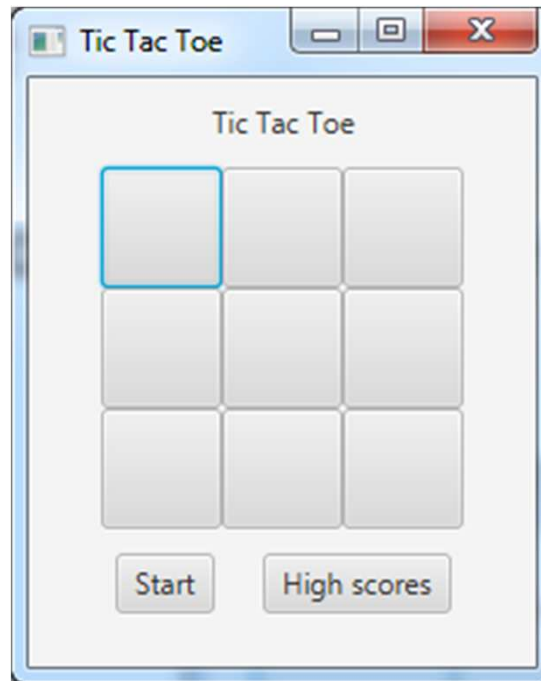
Inspecting JavaFX Applications

- Scenic View is an application for inspecting other running JavaFX applications



Exercise

- Build the following interface (do not implement interaction)



Watch this videos before next session

- Editing tools in NetBeans
 - <http://politube.upv.es/play.php?vid=68688>
- Exporting NetBeans projects
 - <http://politube.upv.es/play.php?vid=68666>

References

- JavaFX: Getting Started with JavaFX
 - <http://docs.oracle.com/javase/8/javafx/get-started-tutorial/index.html>
- JavaFX: Working with Layouts in JavaFX
 - <http://docs.oracle.com/javase/8/javafx/layout-tutorial/index.html>
- JavaFX API Documentation:
 - <http://docs.oracle.com/javase/8/javafx/api/toc.htm>
- Ensemble
 - Google “Java SE Development Kit Demos and Samples Downloads”
- Scenic View
 - <http://fxexperience.com/scenic-view/>