CSCI 2120:
Software Design & Development II

UNIT4: UI management

GUI framework

JavaFX Controls: Control containers

Overview

1. Introduction
2. Accordion
3. TitledPane
4. SplitPane
5. TabPane
6. ScrollPane

Introduction

- Package: javafx.scene.control
- The JavaFX User Interface Controls (UI Controls or just Controls) are specialized Nodes in the JavaFX Scene graph especially suited for reuse in many different application contexts.
- They are designed to be highly customizable visually by designers and developers. They are designed to work well with layout systems. Examples of prominent controls include Button, Label, ListView, and TextField.
- Since Controls are Nodes in the scenegraph, they can be freely mixed with Groups, Images, Media, Text, and basic geometric shapes.
- While writing new UI Controls is not trivial, using and styling them is very easy, especially to existing web developers.

Architecture

- Controls follow the classic MVC design pattern. The Control is the "model". It contains both the state and the functions which manipulate that state.

- The Control class itself does not know how it is rendered or what the user interaction is. These tasks are delegated to the Skin ("view"), which may internally separate out the view and controller functionality into separate classes, although at present there is no public API for the "controller" aspect.

- All Controls extend from the Control class, which is in turn a Parent node, and which is a Node. Every Control has a reference to a single Skin, which is the view implementation for the Control.

- The Control delegates to the Skin the responsibility of computing the min, max, and pref sizes of the Control, the baseline offset, and hit testing (containment and intersection). It is also the responsibility of the Skin, or a delegate of the Skin, to implement and respond to all relevant key events which occur on the Control when it contains the focus.

Control class

- Control extends from Parent, and as such, is not a leaf node. From the perspective of a developer or designer the Control can be thought of as if it were a leaf node in many cases. For example, the developer or designer can consider a Button as if it were a Rectangle or other simple leaf node.

- Since a Control is resizable, a Control will be **auto-sized to its preferred size** on each scenegraph pulse. Setting the width and height of the Control does not affect its preferred size. When used in a layout container, the layout constraints imposed upon the Control (or manually specified on the Control) will determine how it is positioned and sized.

- The Skin of a Control can be changed at any time. Doing so will mark the Control as needing to be laid out since changing the Skin likely has changed the preferred size of the Control. If no Skin is specified at the time that the Control is created, then a default CSS-based skin will be provided for all of the built-in Controls.

Control class

- Each Control may have an optional tooltip specified. The Tooltip is a Control which displays some (usually textual) information about the control to the user when the mouse hovers over the Control from some period of time. It can be styled from CSS the same as with other Controls.

- focusTraversable is overridden in Control to be true by default, whereas with Node it is false by default. Controls which should not be focusable by default (such as Label) override this to be false.

- The getMinWidth, getMinHeight, getPrefWidth, getPrefHeight, getMaxWidth, and getMaxHeight functions are delegated directly to the Skin. The baselineOffset method is delegated to the node of the skin. It is not recommended that subclasses alter these delegations.

Control class declaration

Base class for all user interface controls. A "Control" is a node in the scene graph which can be manipulated by the user. Controls provide additional variables and behaviors beyond those of Node to support common user interactions in a manner which is consistent and predictable for the user.

```
public abstract class Control extends Parent implements Skinnable
```

Additionally, controls support explicit skinning to make it easy to leverage the functionality of a control while customizing its appearance.

See specific Control subclasses for information on how to use individual types of controls.

Most controls have their focus Traversable property set to true by default, however read-only controls such as Label and ProgressIndicator, and some controls that are containers ScrollPane and ToolBar do not. Consult individual control documentation for details.

Styling Controls

- There are two methods for customizing the look of a Control. The most difficult and yet most flexible approach is to write a new Skin for the Control which precisely implements the visuals which you desire for the Control. Consult the Skin documentation for more details.

- The easiest and yet very powerful method for styling the built in Controls is by using CSS. Please note that in this release the following CSS description applies only to the default Skins provided for the built in Controls. Subsequent releases will make this generally available for any custom third party Controls which desire to take advantage of these CSS capabilities.

- Each of the default Skins for the built in Controls is comprised of multiple individually styleable areas or regions. This is much like an HTML page which is made up of 's and then styled from CSS. Each individual region may be drawn with backgrounds, borders, images, padding, margins, and so on. The JavaFX CSS support includes the ability to have multiple backgrounds and borders, and to derive colors. These capabilities make it extremely easy to alter the look of Controls in JavaFX from CSS.

Styling Controls

- The colors used for drawing the default Skins of the built in Controls are all derived from a base color, an accent color and a background color. Simply by modifying the base color for a Control you can alter the derived gradients and create Buttons or other Controls which visually fit in with the default Skins but visually stand out.

- As with all other Nodes in the scenegraph, Controls can be styled by using an external stylesheet, or by specifying the style directly on the Control. Although for examples it is easier to express and understand by specifying the style directly on the Node, it is recommended to use an external stylesheet and use either the styleClass or id of the Control, just as you would use the "class" or id of an HTML element with HTML CSS.

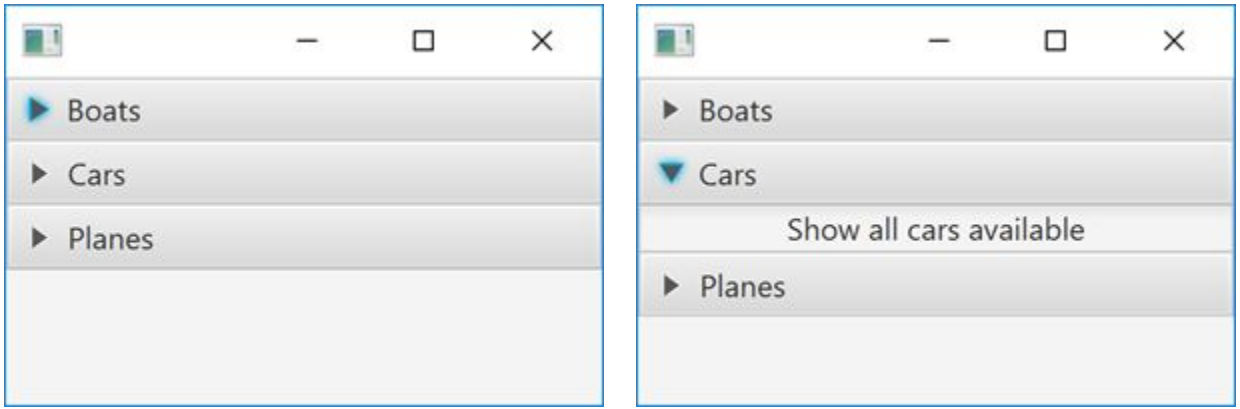
- Each UI Control specifies a styleClass which may be used to style controls from an external stylesheet. For example, the Button control is given the "button" CSS style class. The CSS style class names are hyphen-separated lower case as opposed to camel case, otherwise, they are exactly the same. For example, Button is "button", RadioButton is "radio-button", Tooltip is "tooltip" and so on.

- The class documentation for each Control defines the default Skin regions which can be styled. For further information regarding the CSS capabilities provided with JavaFX, see the CSS Reference Guide.

Control container classes

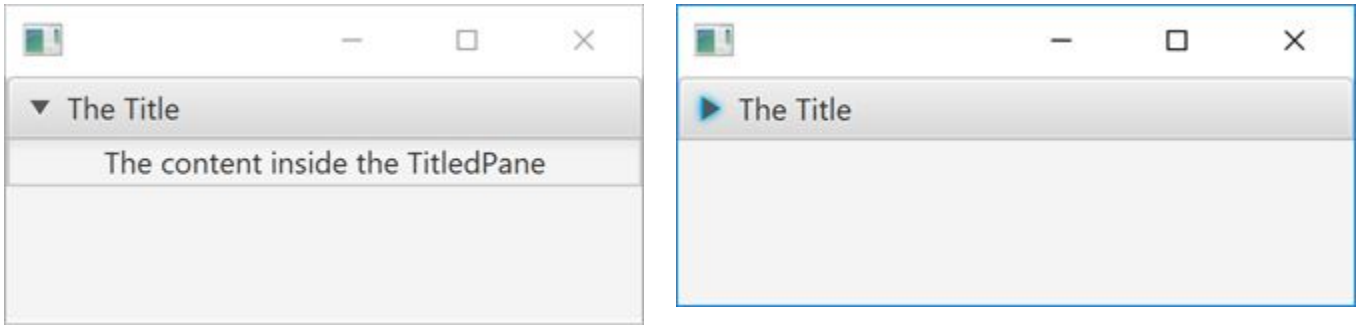
Accordion → Introduction

Accordion


Boats
Cars
Planes
Boats
Cars
Show all cars available
Planes

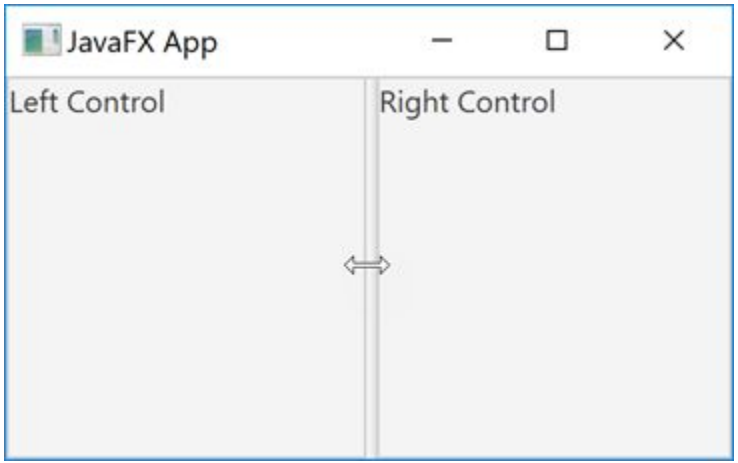
TitledPane → Introduction

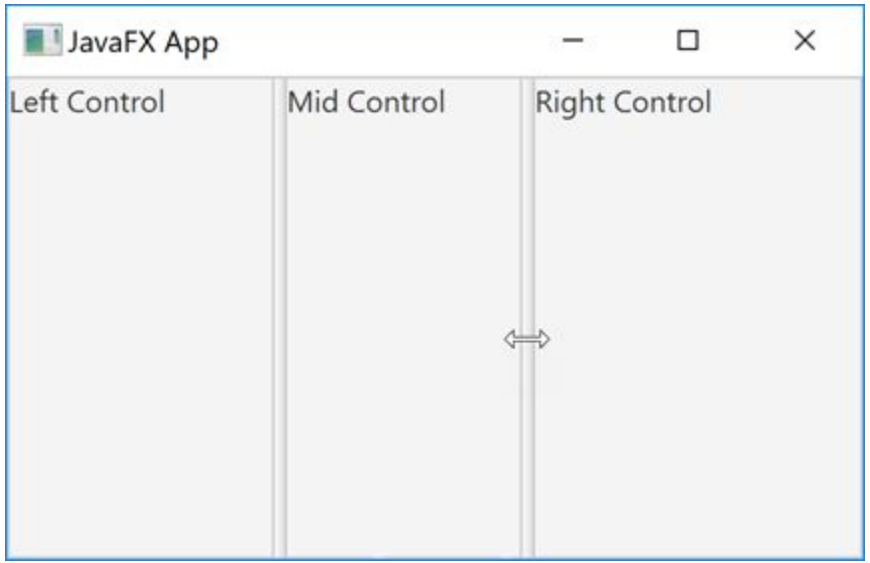
TitledPane



SplitPane → Introduction

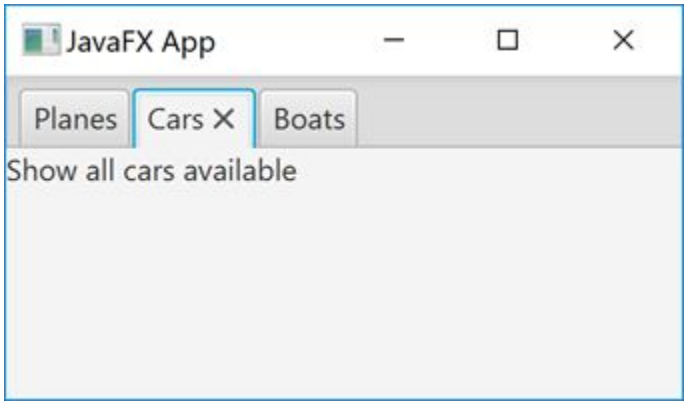
SplitPane


JavaFX App
Left Control
Right Control


JavaFX App
Left Control
Mid Control
Right Control

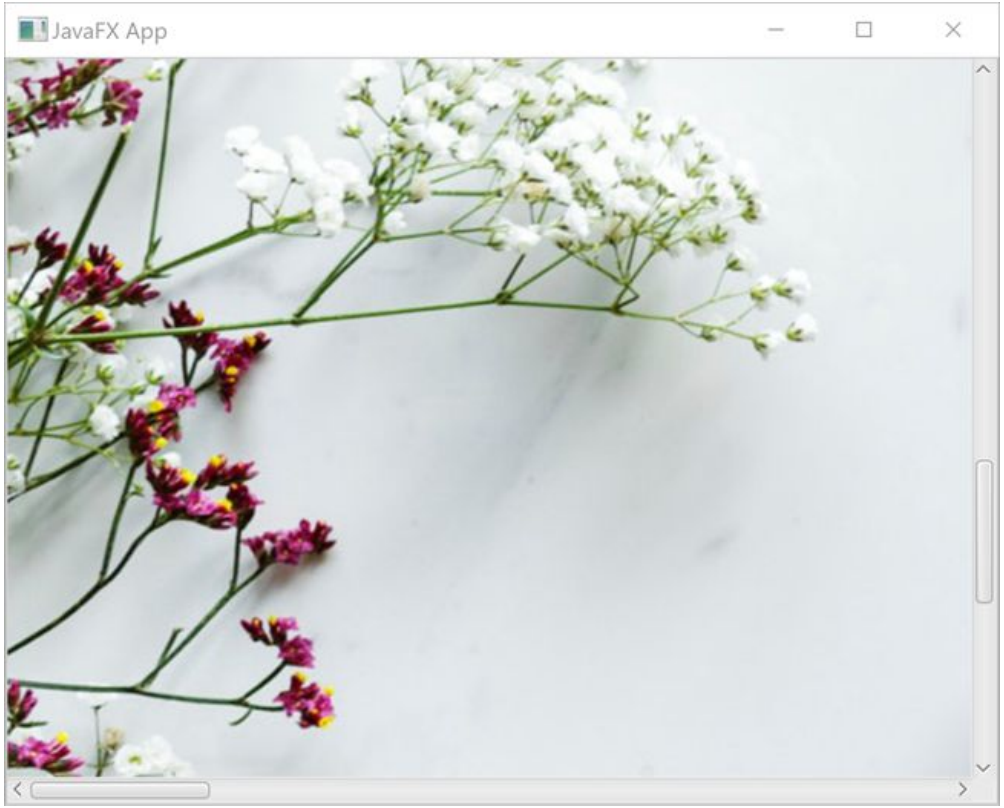
TabPane → Introduction

TabPane



ScrollPane → Introduction

ScrollPane



END