
JavaOne™

Custom Controls

Paru Somashekar
parvathi.somashekar@oracle.com

Jonathan Giles
Tech Lead, JavaFX UI Controls
jonathan.giles@oracle.com

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

- JavaFX UI Controls
  - Definition of a UI Control
  - Options for developing Controls
- Worked examples
- Looking ahead

JavaFX UI Control

- A Control might be loosely defined as
  - A JavaFX Node that allows for user interaction,
  - It does not necessarily extend from Control,
  - It is not necessarily reusable
  - It is commonly styled using CSS

Options for developing Custom UI controls

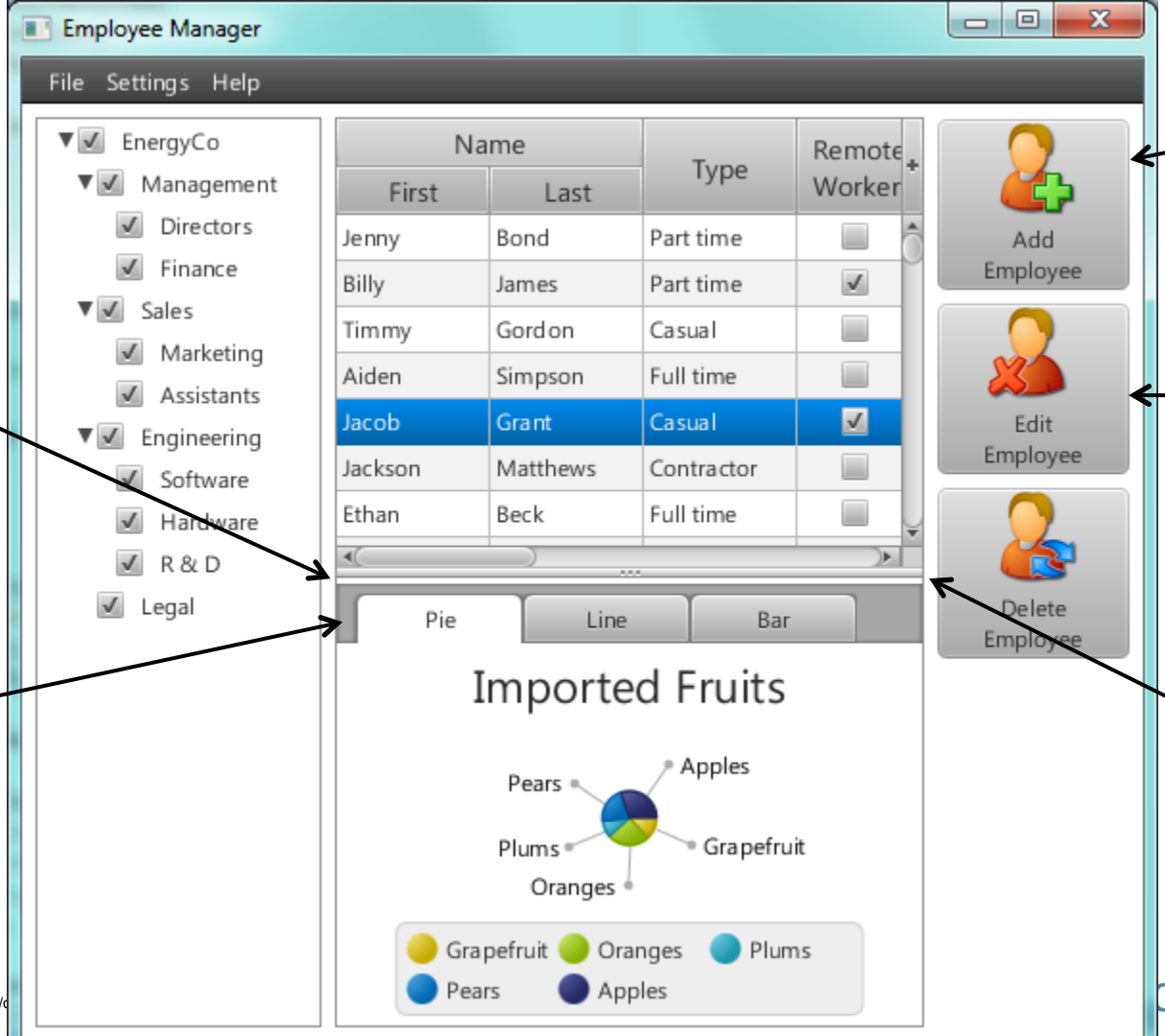
- Simplest
  - Use custom CSS to style an existing control
- For Custom / Unique functionality
  - Extend layout container and use custom CSS
- For redistribution as a library
  - Extend Control and / or create a new Skin, use custom CSS

Option 1: CSS styling

- All JavaFX 2.x controls are styled via CSS
- All styling is contained within caspian.css file
- You can overwrite any of this styling for your own purposes
  - Specify shapes
  - Fill and Stroke
  - Specify effects etc

SplitPane shows a border around all content.

Colour behind tabs differs from other gradients.



UI styled to look sharp / square: default round buttons look out of place.

Border colour of buttons is a lighter grey than the other borders.

SplitPane divider looks out of place.

Custom CSS: TabPane

```
// make border around TabPane be darker (by default the border is hidden)
.tab-content-area {
    -fx-border-color: gray;
    -fx-border-width: 0 1 1 1;
}

// Change background colour behind tabs to have a nicer gradient fill
.tab-pane *.tab-header-background {
    -fx-background-color: -fx-shadow-highlight-color, -fx-outer-border, -fx-inner-border, -fx-body-color;
    -fx-background-insets: 0 0 -1 0, 0, 1, 2;
}
```

Custom CSS: SplitPane

```
// Hide the border around the SplitPane
.split-pane {
    -fx-background-insets: 0;
}

// Hide the SplitPane divider (although retain the ability to drag with mouse)
.split-pane *.split-pane-divider {
    -fx-opacity: 0;
}

// Adding padding around TableView, for special case when embedded within SplitPane
.split-pane > * > * > .table-view { -fx-padding: 1px; }
```

Custom CSS: Button

```
// Make the outer border a darker gray, and remove rounded corners
.button {
    -fx-background-color: -fx-shadow-highlight-color, gray, -fx-inner-border, -fx-body-color;
    -fx-background-radius: 0;
}
```

Using custom stylesheets

- Define custom stylesheet and override default
- Stylesheets are applied to nodes in a scene

Using custom stylesheets

```
@Override public void start(Stage primaryStage) throws Exception {
      primaryStage.setTitle("Employee Manager");

       BorderPane root = new BorderPane();
       …
       Scene scene = new Scene(root);

       String javaoneCSS = getClass().getResource("javaone.css").toExternalForm();
       scene.getStylesheets().add(javaoneCSS);

       primaryStage.setScene(scene);
       primaryStage.show();
 }
```

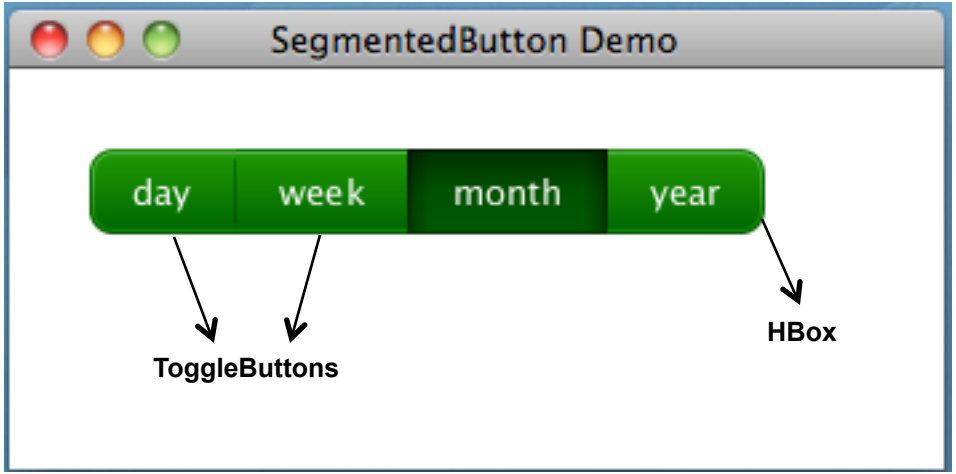
Option 2: Unique functionality

- No need to extend from Control
  - need more functionality than provided by a existing control.
  - not necessary to provide as a library control
- Extend Layout container and CSS styles
- Upsides to this approach:
  - Far less overhead (no need for separate Skin class, lower Node count, etc)
- Downsides:
  - Slightly more difficult to reuse control (must manually install CSS styling into scene)

```
scene.getStyleSheets().add("pathtostylesheet");
```

Option 2: Segmented Button example

- Horizontal Button made of multiple segments.
- Extends HBox
- Adds ToggleButtons to the container.
- Applies additional styles and custom CSS to these buttons



Option 2: Segmented Button Code

```
public SegmentedButton extends HBox  {
   private ObjectProperty<ObservableList<ToggleButton>> buttons;

   public SegmentedButton(ObservableList<ToggleButton> buttons) {
    getStyleClass().add("segmented-button");
    setButtons(buttons);
    buttons.addListener(new InvalidationListener() {
        @Override public void invalidated(Observable o) {
            updateButtons();
        }
    });
 }
```

Option 2: Segmented Button Code

```
private void updateButtons()  {
      for (int i = 0; i < getButtons().size(); i++) {
         ToggleButton t = getButtons().get(i);
         t.setToggleGroup(group);
         getChildren().add(t);
         if (i == buttons.get().size() - 1) {
             t.getStyleClass().add((i == 0) ? "only-button" : "last-button");
         } else if (i == 0) {
             t.getStyleClass().add("first-button");
         } else {
             t.getStyleClass().add("middle-button");
         }
    }
}
```

Option 2: Segmented Button CSS

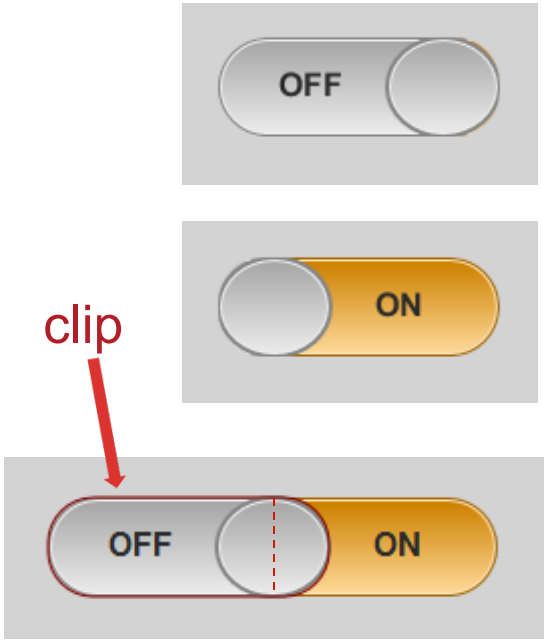
```
.segmented-button .first-button {
        -fx-background-radius: 12 0 0 12;
        -fx-background-insets: 0 0 -1 0, 0, 1 0 1 1, 2 0 2 2;
        -fx-border-insets: 4 0 4 0;
}
.segmented-button .last-button {
        -fx-background-radius: 0 12 12 0;
        -fx-background-insets: 0 0 -1 0, 0, 1 1 1 0, 2 2 2 0;
        -fx-border-insets: 4 0 4 0;
}
.segmented-button .middle-button {
        -fx-background-radius: 0;
        -fx-background-insets: 0 0 -1 0, 0, 1 0 1 0, 2 0 2 0;
        -fx-border-insets: 4 0 4 0;
}
```

Option 2: Segmented Button CSS

```
.segmented-button .toggle-button:selected  {
    -fx-background-color:  -fx-shadow-highlight-color,
            linear-gradient( to bottom, derive(-fx-color,-90%) 0%, derive(-fx-color,-60%) 100% ),
            linear-gradient( to bottom, derive(-fx-color,-60%) 0%, derive(-fx-color,-35%) 50%, derive(-fx-color,-30%) 98%, derive(-fx-color,-50%) 100% ),
            linear-gradient( to right, rgba(0,0,0,0.3) 0%, rgba(0,0,0,0) 10%, rgba(0,0,0,0) 90%, rgba(0,0,0,0.3) 100% );
    -fx-background-insets: 0 0 -1 0, 0, 1, 1;
    -fx-text-fill: -fx-light-text-color;
}
```

Option 2: (Example 2) ToggleSwitch

- Extends Region.
- Can be either On or Off.
- Has two labels (on/off) and a StackPane in the center for the slider.
- Animates the state change.
- Sets a clip to show only one of the state at any given time.



Option 2: ToggleSwitch Code

```
public class ToggleSwitch extends Region {
   final Label offLabel = new Label("Off");
   final Label onLabel = new Label("On");
   final StackPane slider = new StackPane();
   Timeline timeline = new Timeline();

   public ToggleSwitch() {
        getChildren().addAll(offLabel, onLabel, slider);
        // clipRect is a rectangle whose width is set the size of half of the Region's Width.
        setClip(clipRect);
        // add event handlers for interaction
   }
```

Option 2: ToggleSwitch Code

```
/** animates when either the slider or ON/OFF label is clicked
  clip rectangle moves in the opposite direction of the Region itself.
*/
private void animate() {
     timeline.getKeyFrames().addAll (
          new KeyFrame(Duration.ZERO, new KeyValue(clipRect.xProperty(), 0)),
          new KeyFrame(Duration.ZERO, new KeyValue(layoutXProperty(), 0)),
          new KeyFrame(new Duration(400), new KeyValue(clipRect.xProperty(), movingWidth),
                                new KeyValue(layoutXProperty(), -movingWidth)));
     timeline.setRate((!toggle.get()) ? 1 : -1);
     timeline.play();
     toggle.set(!toggle.get()); // used to reverse animation direction.
 }
```

Specifies the direction of animation

Option 2: ToggleSwitch CSS

```
.toggle-switch {
      -fx-background-color: derive(-fx-color,-36%), derive(-fx-color,73%),
        linear-gradient(to bottom, derive(-fx-color,-19%),derive(-fx-color,61%));
      -fx-background-radius: 30;
}
.on-label {
       -fx-background-color: derive(orange,-36%), derive(orange,73%),
             linear-gradient(to bottom, derive(orange,-19%),derive(orange,61%));
      -fx-background-radius: 0 30 30 0;
}
.off-label {
       -fx-background-radius: 30 0 0 30;
}
.on-off-slider {
        -fx-background-radius: 30;
}
```

ToggleSwitch as CheckBoxSkin

- ToggleSwitch has the same functionality as that of a CheckBox.
- Implement ToggleSwitchSkin that implements the Skin Interface.
- Add “toggle-switch” styleclass to CheckBox.

```
checkbox.getStyleClass().setAll(“toggle-switch);
```

- Swap in ToggleSwitchSkin for CheckBox by specifying (in CSS):

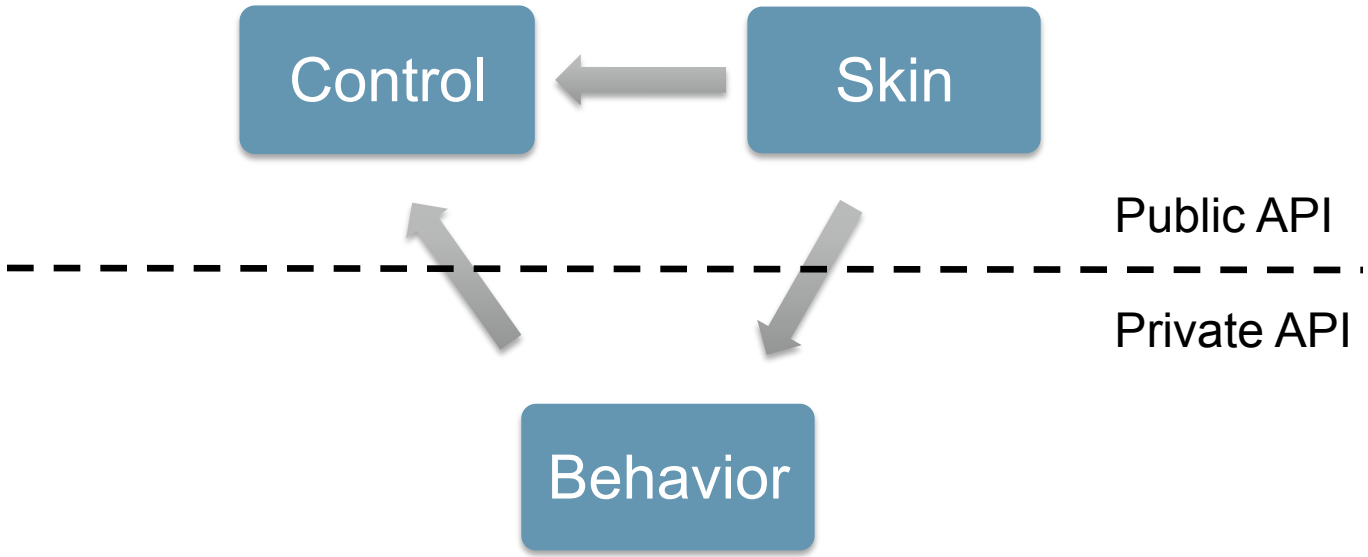
```
.toggle-switch {
    -fx-skin: <path to ToggleSwitchSkin>;

}
```

Option 3: For redistribution of controls

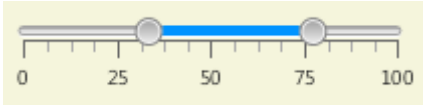
- Extend the Control class and or provide custom skin
- All controls that ship with JavaFX 2.x extend Control
- Control is-a Node
  - Has state values and all scenegraph transformations are valid
- Controls has-a Skin
  - Visual appearance of Control (has nodes)
- Behavior
  - Interaction like event handling.

Option 3: The Control API



First steps to creating a Custom Control

- Create a class that implements Skin (RangeSliderSkin)
- Create a CSS file to contain your CSS styling (range-slider.css)
  - Create a .range-slider{ … } section, and specify –fx-skin: “com.widgets.RangeSliderSkin”;
- Create a class that extends Control (RangeSlider)
  - Set a style class by calling getStyleClasses().add(“range-slider”)
  - Override getUserAgentStyleSheet and return the path to range-slider.css



- Next steps:
  - Add relevant API into RangeSlider
  - Implement RangeSliderSkin to return visual representation of control, and observe changes in RangeSlider

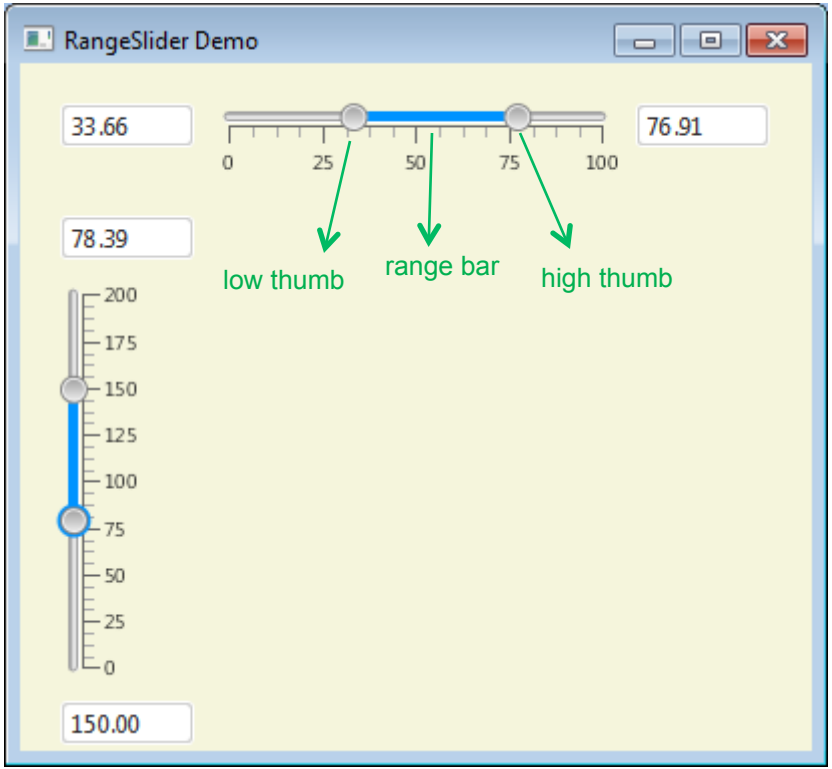
Option 3: RangeSlider Example

Options

- Reuse some of the code from Slider control, skin and behavior
  - Not a good idea to subclass Slider (API would not make sense)
  - Start by **copying the source** into a RangeSlider class.
- RangeSlider extends **Control** class
- RangeSliderSkin implements **Skin** Interface
- RangeSliderBehavior to handle slider interaction.
- rangeslider.css contains **CSS** styling

Option 3: RangeSlider Properties

```
public class RangeSlider extends Control {
  private DoubleProperty lowValue;
  private BooleanProperty lowValueChanging;
  private DoubleProperty highValue;
  private BooleanProperty highValueChanging;
  ...and other properties of Slider.
  public RangeSlider() {
     getStyleClass().add("range-slider");
  }
  @Override
  protected String getUserAgentStylesheet() {
        // "rangeslider.css";
  }
}
```



Option 3: RangeSliderSkin and behavior

```
public class RangeSliderSkin extends SkinBase<RangeSlider, RangeSliderBehavior> {
     // StackPanes used for track, lowThumb, highThumb and rangeBar
     public RangeSliderSkin() {
        lowThumb.getStyleClass().add("low-thumb");
        highThumb.getStyleClass().add("high-thumb");
        rangeBar.getStyleClass().add("range-bar");
        track.getStyleClass().add("track");
     }
}
public class RangeSliderBehavior extends BehaviorBase<RangeSlider> {
  // handles RangeSlider interaction.
}
```

Option 3: rangeslider.css

```
.range-slider > .low-thumb, .range-slider > .high-thumb {
      -fx-background-color: derive(-fx-color,-36%), derive(-fx-color,73%),
       linear-gradient(to bottom, derive(-fx-color,-19%),derive(-fx-color,61%));
       -fx-background-insets: 0, 1, 2;
       -fx-background-radius: 1.0em; /* makes sure this remains circular */
       -fx-padding: 0.583333em; /* 7 */
}
.range-slider > .range-bar {
       -fx-background-color: -fx-focus-color;
}
.range-slider:vertical > .track {
       -fx-background-color: -fx-shadow-highlight-color, derive(-fx-color,-22%),
        linear-gradient(to right, derive(-fx-color,-15.5%), derive(-fx-color,34%) 30%, derive(-fx-color,68%));
       -fx-background-insets: 0 -1 0 1, 0, 1;
}
```

Option 3: (Example 2) Rating control

```
public class Rating extends Control {
     // public properties
     IntegerProperty rating;
     IntegerProperty max;
     ObjectProperty<Orientation> orientation;

     @Override protected String getUserAgentStylesheet() {
         return getClass().getResource("rating.css").
              toExternalForm();
     }
}
```

Option 3: (Example 2) Rating control

```
public class RatingSkin extends SkinBase<Rating, RatingBehavior> {
     private static final String STRONG = "strong";
     public RatingSkin(Rating control) {
        Pane btnContainer = isVertical() ? new VBox() : new HBox();
        // Add ToggleButtons to the pane based on max value.
     }
         private void updateRating(int newRating) {
        // for all toggle in buttons
        toggle.getStyleClass().remove(STRONG);
              if (toggleIndex <= newRating) {
                   toggle.getStyleClass().add(STRONG);
              }
     }
}
```

Option 3: (Example 2) Rating control CSS

```
.rating > .container > .toggle-button {
    -fx-background-color: transparent;
    -fx-background-image: url("unselected-star.png");
    -fx-padding: 8 16;
    -fx-background-image-repeat: no-repeat;
}
.rating > .container > .toggle-button.strong {
    -fx-background-image: url("selected-star.png");
}
.rating > .container > .toggle-button:hover {
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 8, 0.0 , 0 , 0 );
}
```

default style

Worked Examples

Looking Ahead

Looking Ahead

- There is a lot that is planned for JavaFX 8.0 (for UI controls):
  - Public API for custom UI controls
  - Public (Java-based) API for CSS
  - RTL / bidi text support
  - A11y support
  - TreeTableView (maybe)
  - Charts improvements
  - Miscellaneous features / tweaks
  - Performance improvements
  - Bug fixes

Looking Ahead

Public API for custom UI controls

- In JavaFX 2.2, Control/Skin is public API
  - However, the base Skin implementation, SkinBase, has always been private API
  - This makes building 3rd party controls more difficult / risky than it needs to be
  - However, almost all 3rd party controls use SkinBase rather than Skin
- The controls API will be made public and finally committed to for JavaFX 8.0
- Already there are two changes and ideas that we can talk about…

Looking Ahead

Public API for custom UI controls

http://javafx-jira.kenai.com/browse/RT-21596

Looking Ahead

Public API for custom UI controls

- Change #1: SkinBase no longer extends from Region
  - Pros:
    - Controls are lighter-weight: Control extends from Region, SkinBase extends from Object
    - Controls are now conceptually simpler: a Control contains children which the Skin lays out (previously the controls children were separate from the skins children, which was confusing)
  - Cons:
    - All skins extending from SkinBase (naughty people!) are now broken

- SkinBase, in JavaFX 8.0, has moved to be public API (although the API isn't finished changing quite yet)

Looking Ahead

Public API for custom UI controls

- (Proposed) Change #2: Behavior does not move into public API, introduce alternative public API for recording key / mouse actions

- Current in-research approach uses annotations to annotate methods, and input/action maps on the control to store these.

Looking Ahead

Public API for custom UI controls

- You would annotate a method as shown below, and most probably group together all methods relevant to a single control in a single class.
  - In other words, the Behavior classes for our controls are perfect for this approach – they will just become implementation details rather than public API.

```
@ActionMethod(actionCommands={
    @KeyEventMethod(keyCode="space", keyEventType=KEY_PRESSED),

    // Mac OS doesn't support enter press, so we add Enter support for
    // everyone apart from Mac OS
    @KeyEventMethod(keyCode="enter", keyEventType=KEY_PRESSED, mac=false),
})
public void keyPressed() {
    …
}
```

Looking Ahead

Public API for custom UI controls

- Then, rather than have the Skin defer all key interactions to the behavior, the constructor call the ActionManager to parse the annotations from a given object.

```
// initialise behavior by populating the button input and action maps
ActionManager.populateControlMaps(button, new ButtonBehavior<Button>(button));
```

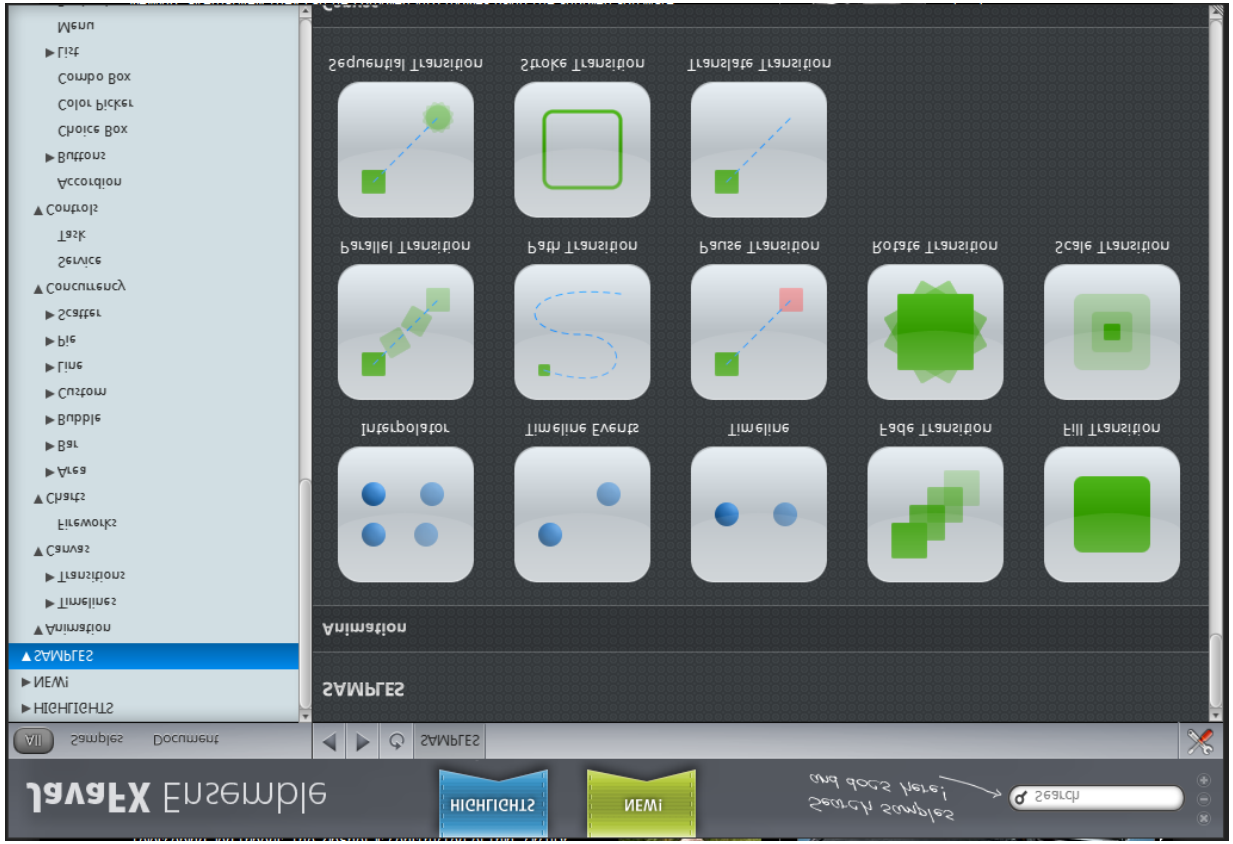
- What this does is populate two new maps in the control (in this case Button):

```
ObservableMap<KeyCombination, Object> inputMap;
ObservableMap<Object, Action> actionMap;
```

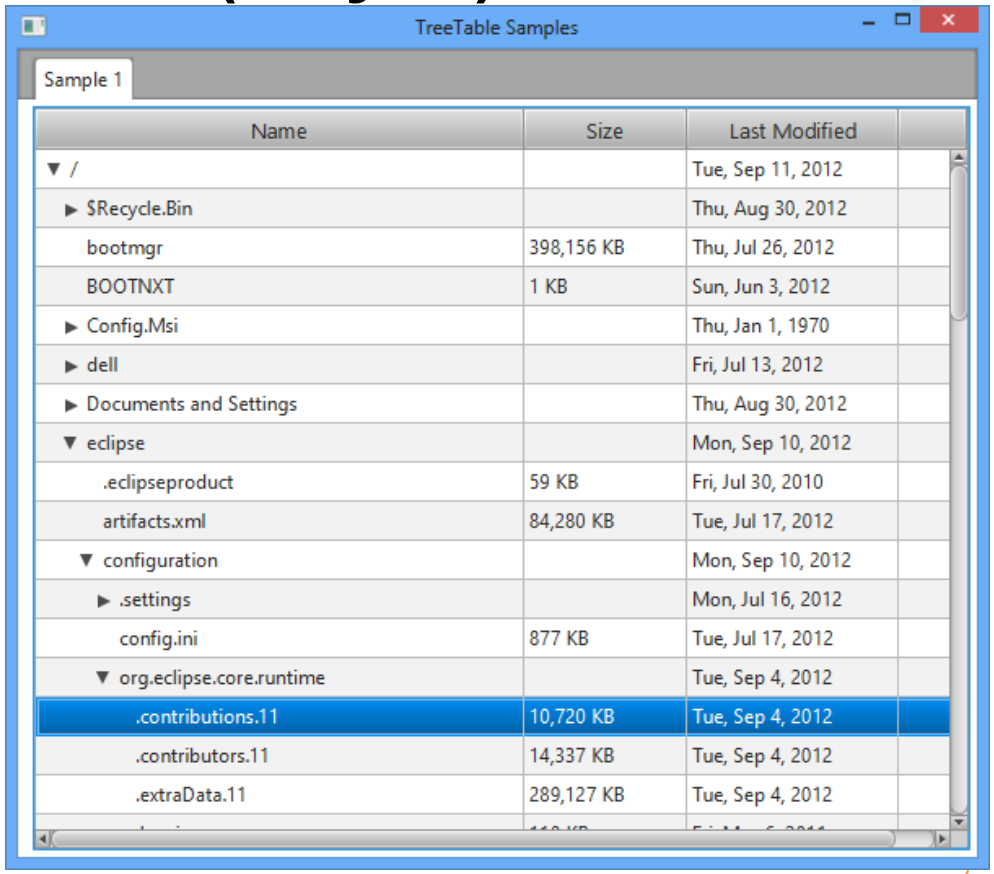
- These will be called (via SkinBase) when a key event occurs
- These two maps can then be manually manipulated by people wanting to modify behaviors

RTL Support

RTL Support



TreeTableView (maybe)



Accessibility Support

- Accessibility support on Windows and Mac platform.
  - Using standard accessibility interfaces
    - UIAutomation on Windows
    - NSAccessibility on Mac
- Full keyboard support for navigation
- Better support for high contrast


JavaOne™
Thanks for attending!
Any Questions?
Paru Somashekar
parvathi.somashekar@oracle.com
Jonathan Giles
Tech Lead, JavaFX UI Controls
Jonathan.giles@oracle.com
@JonathanGiles
MAKE THE FUTURE JAVA
ORACLE

MAKE THE FUTURE JAVA

ORACLE


JavaOne™