Creating custom widgets in JavaFX

Creating Custom Widgets in JavaFX

- One of the core skills of any GUI developer is the ability to create your own custom widgets
  - Requires you to design an interface
  - Interpret input events from keyboard, mouse and touch
  - Based on this you will perform some action, adjust your widget, fire some events etc.

Creating Custom Widgets in JavaFX

- Implementing a custom widget requires you to understand two core elements
  - How widgets are drawn on screen
  - How to catch and interact with events

Creating Custom Widgets in JavaFX

- We will start with the catching and interaction of events
- There are three main classes of input events in JavaFX
  - Mouse events
  - Keyboard events
  - Touch events

Creating Custom Widgets in JavaFX

- The handlers for all three events are defined in a similar way to event handlers for ActionEvents
  - Except instead of an ActionEvent you will be using a KeyEvent for keyboard events. MouseEvent for mouse events and DragEvent for mouse drag events
  - You will be given one of three event types for keyboard input
    - Key pressed: a key has been pressed and held down
    - Key released: a previously pressed down key has now been released
    - Key typed: a key that has been pressed and released quickly

Creating Custom Widgets in JavaFX

- To handle a key event you are required to determine what key has been pressed and check if it is relevant to your widget
  - If not discard it
  - Otherwise perform an action

Creating Custom Widgets in JavaFX

- Mouse events and touch events are similar
- Mouse events have a single pointer and one of these four event types
  - MousePressed: a button has been pressed and held down
  - MouseReleased: a button that was held down has now been released
  - MouseMoved: while a button has been held down the mouse has been moved
  - MouseClicked: a button has been clicked and released in quick succession.

Creating Custom Widgets in JavaFX

- Mouse events also contain information about the location of the pointer relative to the top left corner of the widget
  - The top left corner is considered the origin of the widget with coordinates (0,0)
  - X axis increases as you go further right
  - Y axis increases as you go further down
    - Both values are measured in pixels and will be between [0, width) for X
    - [0, height) for Y

Creating Custom Widgets in JavaFX

- For both key and mouse events there may be modifier keys active
- Modifier keys are one of the following
  - Control
  - Alt
  - Shift

Creating Custom Widgets in JavaFX

- Touch events are similar to mouse events but differ in two crucial respects
  - They are not as precise
    - A pointer has single pixel accuracy
    - Whereas touch represents an area where the user has made contact with the screen
  - There is the possibility of multi touch
    - Enables gestures
    - More complex control

Creating Custom Widgets in JavaFX

- That covers the event side of things
  - Now we need to cover the drawing side
  - Note that you will require a fair amount of math to draw things accurately
  - Only method we can use to define drawings in any display.

Creating Custom Widgets in JavaFX

- Most GUI toolkits will provide you with a means of drawing on screen
- But it is entirely dependant on the toolkit and its rendering mode
- There are two types of rendering mode with regards to computer graphics
  - Immediate mode
  - Retained mode

Creating Custom Widgets in JavaFX

- Immediate mode is used in a lot of toolkits as it is the easiest to define and interact with
  - In immediate mode as soon as you issue a drawing command it is sent from CPU across the PCI-Express bus wherein it is rendered by the GPU immediately
  - Simplifies rendering
  - But at the cost of performance

Creating Custom Widgets in JavaFX

- Retained mode rendering is what JavaFX uses and is the standard method for videogames.
  - It will store as much information as possible in GPU memory
  - Only issues rendering commands for the GPU to redraw items
  - GPU memory is an order of magnitude faster than main memory
    - Eliminates bus transfers as well
    - More information is rendered in the same time
  - However, it takes more effort and code to setup

Creating Custom Widgets in JavaFX

- JavaFX implements retained mode rendering by using a scenegraph approach
- Where each node in the graph represents an object to be rendered
- In fact up to this point you have been working with the scene graph directly
  - Where each layout and control has been a node in the scene graph

Creating Custom Widgets in JavaFX

- When you are building custom widgets and doing custom drawing you will mostly be working with primitive objects
  - Lines, rectangles, circles, ellipses, etc
- And transformations
  - Scale, rotate, translate

Creating Custom Widgets in JavaFX

- Every object in the scenegraph is composed of primitives and transformations
  - The reason for this is to cut down on the amount of information that is transmitted from CPU to GPU.
  - If an object is going to have the same shape and size for the majority of its existence then there is no point resending that information over the bus every time you wish to adjust its position, scale or rotation
    - There are potentially 60 renders a second depending on the activity in your application
    - Approximately 16ms for each render

Creating Custom Widgets in JavaFX

- Thus the transforms state that an object should be moved to a certain position, and rotated or scaled as necessary.
- Thus if you want to adjust the position of a rectangle you only need to apply a new translation object.
  - In 2D this would be two coordinates (x,y)
  - Instead of doing a full calculation to compute the new top left and bottom right corners of the rectangle
  - And sending that information across the bus

Creating Custom Widgets in JavaFX

- In all toolkits these transforms are done in the form of square matrices
  - The neat property of this is that when all transforms are multiples together you get a single matrix that is a composition of all the transforms.
- Secondly with the use of a matrix stack it is possible to move objects around the scene
  - While reducing floating point error

Creating Custom Widgets in JavaFX

- The work that goes into building a custom control in JavaFX is badly implemented in JavaFX
  - Could do with some serious improvement
- The JavaFX model requires you to override the general Control class
  - However the documentation states that the Skin class and the Behaviour class should be overridden too.

Creating Custom Widgets in JavaFX

- This is problematic however,
  - Mainly because the Behaviour base class is private and thus near impossible to access
  - Without some over the top hacking to get access to it

Creating Custom Widgets in JavaFX

- Also there is very little documentation on the Skin class
  - How it interacts with the control and where and when various functions are called
- With the lack of documentation on these pieces the best we can do is
  - Subclass the control class to get our widget and its event handling
  - Attach a subclass of the Pane class to get some rendering ability
  - And attach primitives as children of our Pane subclass to make our control

Creating Custom Widgets in JavaFX

- It's long winded but it's the only option we have to generate custom controls
  - In the absence of better documentation
- Drawing in JavaFX requires you to override one method in the subclass of your pane class
  - Resize() which takes in a new width and height for the size of your pane
    - Gives you a chance to reposition and resize everything

Creating Custom Widgets in JavaFX

- After this you will need to use primitives such as lines, circles and text
  - These should be children of your Pane class
  - And then resized and repositioned by the overridden resize() method of your Pane class
- Note that you should have transforms attached to each of your children
  - Instead of absolute coordinates
  - To minimise information sent over the bus

Creating Custom Widgets in JavaFX

- If you have a group of primitives that represent a move complex object that forms part of your view that is generated by the Pane
  - You should subclass the group class
    - Group itself renders nothing
    - Is a collection of related children that should move and stick together

Creating Custom Widgets in JavaFX

- To make a group class you need to subclass the Group class
  - And override the following methods
    - resize() in case the Pane wants to make this Group larger or smaller
    - relocate() in case the Pane wants to move this Group to a different position