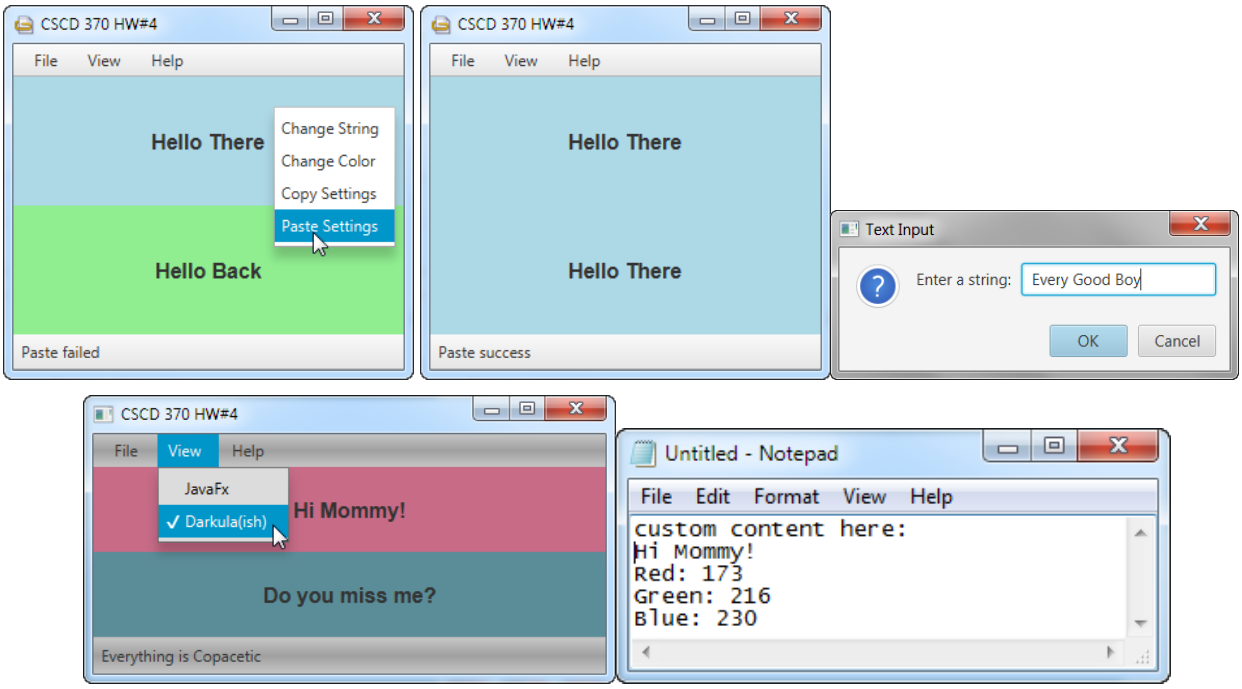
CSCD 370: GUI Programming with JavaFX
HW 3: Clipboard, CSS

In this assignment you will build a simple demonstration of clipboard transfers along with application skinning using a simple CSS stylesheet. In this case I provide specifications and some hints instead of a step-by-step guide. In the specifications (below) I refer to the two main portions of the screen as "panels." This is NOT meant to imply that they are built from the JavaFX Panel class. The hints section tells you how I implemented those panels, but do it however you like.



Specifications

1) Your solution should be based on the general template from Lab 0, and have an appropriate title and a Help menu with an About Box.
2) The content area should contain two panels that share the available space equitably (horizontally or vertically). Each panel should have a background color and text associated with it. The initial color should not be white or black, and the initial text should not be empty.
3) The subpanels should grow and shrink with the main window, and the text should remain centered in the subpanel as the window is expanded and shrunk.
4) There should be a view menu with two entries that allow the user to select either the default stylesheet or some other custom stylesheet that changes the look of the window without obscuring the contents. The current choice must be shown on the menu.
5) Right-clicking on either subpanel should bring up a context menu with the illustrated options.
6) Selecting Change String should bring up a text input dialog as shown. Don't make that dialog any more complicated than what is shown (i.e., it contains no header area). Pressing OK should replace the text in the center of the subpanel. Pressing Cancel should do nothing other than close the dialog.

7) Selecting Change Color should generate a random color for that panel background. That means random red, green, and blue components, not a random selection from a predetermined set.
8) Selecting Copy Settings should copy that subpanel's text and color to the system clipboard.
9) If the clipboard contains custom data from this application, selecting Paste Settings should replace that subpanel's text and color from the clipboard content.
10) If the clipboard does not contain custom data from this application, but contains a non-blank string, selecting Paste Settings should replace the text of the subpanel with that string.
11) If the clipboard has neither custom application data nor a non-blank string, the status bar should indicate a paste failure, as shown.
12) When the clipboard contains custom data from this application, the user should be able to paste into a text editor and get the text on the subpanel along with a text representation of the color, in RGB units from 0 to 255, as shown above.
13) Your main window should use a custom icon on the title bar.
14) Turn in your java sources, your style sheet, and your jar file.

Hints

1) It is up to you how to implement the subpanels. I chose to create a custom panel class by subclassing Region, which is the base class for all layout containers in JavaFX. Region supports a background, but I found the clipboard work was easier to implement if I represented the background with a Rectangle, which I bound to the dimensions of the Region. The center of my BorderPane is a VBox containing two of those custom panels. If you also do it that way, you might want to take a look at VBox.setVGrow(), which is (a bit oddly) a static method that controls the priority of how children are grown to its available space.
2) Unlike StackPane, Region does not automatically center its child Nodes. In order to make my Region center the text, which I placed in a Label, I overrode layoutChildren(). In that method I get the width and height of my panel, the width and height of the Label, and calculate the placement of the Label from that. I then call relocate() on the Label to place it in the desired position. This is similar to what we did for the seven segment display, but easier.
3) I implemented the context menu, and its associated MouseEvent handler, in my custom panel.
4) There are a couple of ways you can get messages (like paste failures) from your panel class back to your application class. An old-school way is to simply provide a callback in your application class, in which case I recommend that your panel class declare an interface that the "user" class must implement (see the course notes on custom dialogs). You could also create a property in the panel class (see HW #2). The application class could then choose to attach a change listener (or not, if it doesn't care, which is one advantage that properties have over callbacks).
5) My custom panel creates a DataFormat for its custom clipboard data. When handling the copy function, it performs a put on the ClipboardContent using that DataFormat along with an instance of a custom class that encapsulates the text string and RGB values. In order to satisfy requirement 12, it also puts a standard string on the clipboard in case the paste is made to another application (that can accept text from the clipboard).

6) My handler for the paste checks for that custom DataFormat on the clipboard. If it finds it, it extracts and applies the text and color to the panel. If it doesn't find it, it checks to see if there is a String on the clipboard. If it finds a String, it further checks to make sure it is not empty before using it. You'll find the String.trim() method useful here, but may not understand why until you try without it.
7) You may copy the example CSS style sheet from the lecture (or create your own) into your package directory. To revert a style sheet back to its defaults, simply call clear() on the StyleSheet object. Note that applying a style sheet to your Scene does not apply it to any dialog boxes you launch, since those have a separate node tree. You are not required to apply a style sheet to the text editing dialog, but feel free to if you like. Nor is your dialog window required to have a custom icon.
8) Make sure you test that you can make clipboard transfers within your two panels, between instances of your application, and to and from a text editor. On Windows, Netbeans will only let you launch a single instance of your application, so you'll need to navigate to your dist directory to launch a second instance.