CMPT 381 Assignment 1: Widgets, Layout, Events

Due: Friday, Sept. 24, 11:59pm

Overview

In this assignment you will build a user interface to demonstrate your skills with basic JavaFX development (including widgets, events, and layout). The assignment is divided into three parts that have increasing difficulty: part 1 covers simple widget creation and layout; part 2 covers event handling; and part 3 covers some improvements to the look and operation of the UI.

The UI is for a simple JavaFX tool that allows the user to select colours for a three-colour palette and store the palettes in a list. The tool provides sliders for Red, Green, and Blue values, and a scrolling list for storing completed palettes.

The tool will end up looking like the pictures below.

Part 1: Basic widget creation and layout

You should be able to start Part 1 after lecture on Sept. 7, and complete it after lecture on Sept. 9.

Use the following JavaFX widget classes to create the interface shown in the "Part 1 target" picture:

- Circle class (for the top colour swatch)
- Button class (for the "Add to Palette" and "Add to List" buttons)
- Slider class (for the Red, Green, and Blue sliders)
- Label class (for the title labels to the left of the sliders, and the value labels to the right of the sliders)
- ListView<String> (for the list of palettes shown at the right side of the UI)

For laying out the interface, use the following container widget classes:

- HBox (for horizontal arrangements)
- VBox (for vertical arrangements)
- Remember that you can have containers within containers to achieve your desired layout
- Useful methods for adjusting spacing and padding within HBox and VBox include setSpacing() and setPadding()

Notes:

- All widgets and layouts must be created and manipulated programmatically, **not** using FXML
- Your UI does not have to respond to any user events (it just needs to show the widgets and layout)
- Your UI does not have to handle resizing
- Remember that a ListView<String> widget uses an ObservableList<String> object as its abstract model
- You should add a few default strings to the ObservableList so your ListView will show up in the UI

Product of Part 1: a JavaFX project runnable in IDEA.

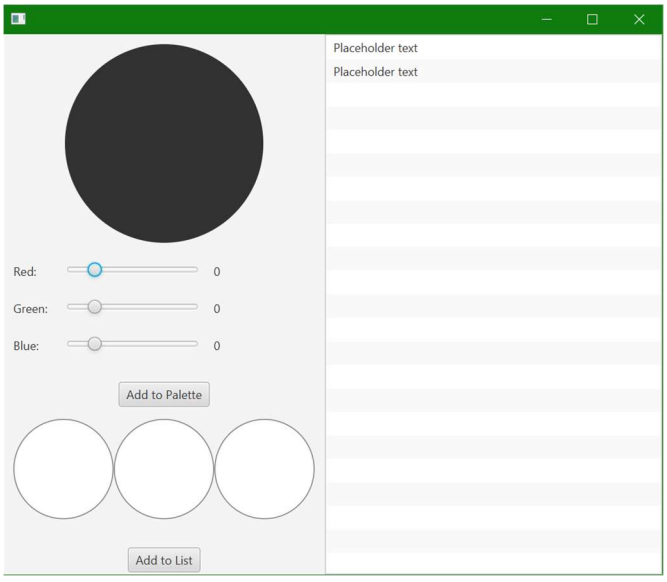

Figure 1: Target of Part 1

Part 2: Events and Interaction

You should be able to complete Part 2 after lecture on Sept. 14.

In Part 2 of the assignment, you will add event handling and make the app interactive. To do this, you will create variables and classes to store information about the current colour and palette, and then add event handling to the widgets in your UI.

First, add code to store information about the current colour and the current palette:

- Add an instance variable Color currentColor to store the current colour as set by the sliders
- Create class ColorPalette that stores three Color objects (which are white by default)
- Create method ColorPalette.addColor(Color c) to add colours to the palette (if all three colours are already assigned, the palette should wrap around to the first colour)

Second, add event handling to the sliders:

- Attach a change listener to the value property of each slider, so that when the slider slides, your method setColor() will be called
- The listener can be coded as a lambda such as `(observable, oldValue, newValue) -> setColor()`
- Method setColor() should read the values of all three sliders, create a Color object using static method Color.rgb(), update the colour of the swatch with method setFill(Color c) of class Circle, and update all of the value labels of the sliders with a number between 0-255

Third, add event handling to the buttons:

- "Add to Palette" button handler: when the user clicks the "Add to Palette" button, the colour that is currently in the swatch should be added to the current palette
- "Add to List" button handler: when the user clicks the "Add to List" button, a String representing the current ColorPalette is added to the ListView's ObservableList
- It is a good idea to write this as a toString() method for class ColorPalette

Notes:

- Set the fill colour of Circle objects using Circle.setFill(), not using CSS (e.g., don't use Circle.setStyle())

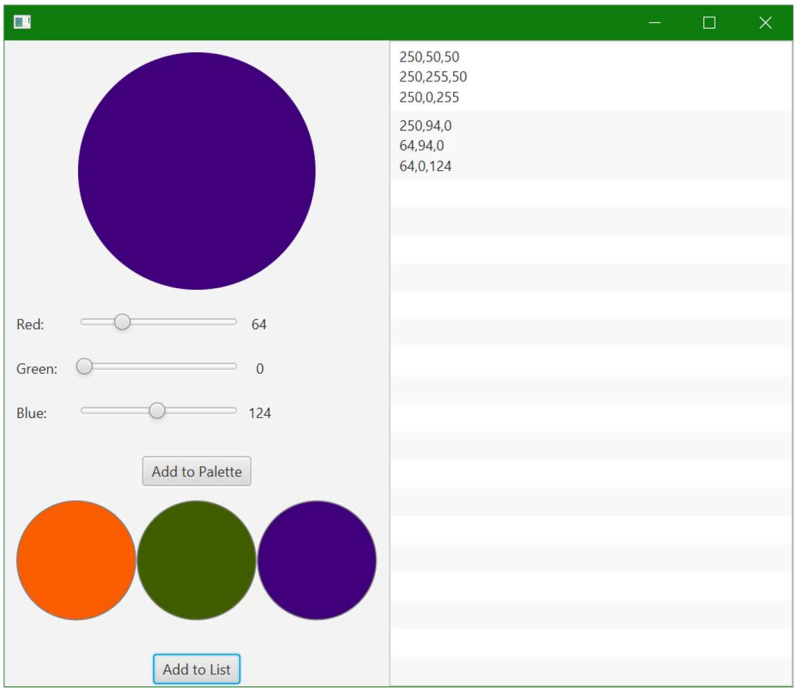

Figure 2. Target of Part 2 (also see video demo).

Product of Part 2: a JavaFX project runnable in IDEA.

Part 3: Enhancements and Generalization

You should be able to complete Part 3 after lecture on Sept. 16.

In Part 3 you will add features to the application such as a custom cell renderer for the ListView, and you will convert aspects of the application to be reusable compound widgets.

First, you will generalize your code for the three colour sliders into a single class, ColorSlider:

- A ColorSlider contains a Label for the title, a Slider, and a Label for the value
- The constructor for ColorSlider should take a string argument to use as the title, and should set up the slider to go from 0 to 255 with an initial value of 50
- Remember that compound widget classes should extend Pane, and should add the top container widget to its own list of children
- The ColorSlider class should have methods getValue() and setValue()
- The ColorSlider class should make the instance variable for the Slider public so that listeners can be attached from the main application (this is a bit of a hack; we will discuss further in lectures)

Second, you will generalize your code for showing the three colours of the palette into a single class, PaletteView:

- A PaletteView contains three Circle objects in an HBox, and also has a reference to a ColorPalette object
- The constructor of PaletteView should take a ColorPalette object as an argument; if the argument is not null it should use the argument as its ColorPalette instance variable; otherwise it should create a new ColorPalette for its instance variable
- PaletteView should extend Pane and add the top container widget to its list of children

Third, you will create a custom cell renderer for showing the actual palettes as the items of the ListView:

- Change your definition of the ListView and ObservableList instance variables to use <ColorPalette> instead of <String>
- Create class PaletteCell that extends ListCell<ColorPalette>
- PaletteCell should have one method, updateItem() that is similar to the following:

```
public void updateItem(ColorPalette item, boolean empty) {
        super.updateItem(item, empty);
        PaletteView pv = new PaletteView(item);
        setGraphic(pv);
    }
```

- When you create your ListView, add a call to method setCellFactory() that is similar to the following:

```
myListView.setCellFactory(listItem -> new PaletteCell());
```

Figure 2. Target of Part 2 (also see video demo).

Product of Part 3: a JavaFX project runnable in IDEA.

What to hand in (note: each student will hand in an assignment)

- Create a zip file of your IDEA project (File → Export → Project to Zip file…). Note that you do not need to hand in separate files for Part 1, 2, and 3: if you have completed Part 2, you do not need to hand in anything for Part 1; if you have completed Part 3, you do not need to hand in anything for Part 1 or 2.
- Add a readme.txt file to the zip that indicates exactly what the marker needs to do to run your code. (Note that systems for 381 should never require the marker to install external libraries, other than JavaFX).

Where to hand in

Hand in your zip file to the Assignment 1 link on the course Canvas site.

Evaluation

Marks will be given for producing a working GUI that correctly uses JavaFX widgets, events, and layout, and that follows the specifications given above. Code should be appropriately documented and tested (although documentation will not be marked). Note that no late assignments will be allowed, and no extensions will be given, without medical reasons.