CMPT 381 Assignment 2: Layout Simulator

Due: Sunday, Oct. 15, 11:59pm

Overview

In this assignment you will build a layout simulator that demonstrates your understanding of the two-pass layout algorithm discussed in class, as well as your skills with retained-mode graphics in JavaFX. The simulator will implement a basic horizontal linear layout container (similar to a JavaFX HBox) and a set of simulated widgets, and will dynamically update the layout as the window is resized.

Part 1: LayoutView, LinearLayout, and SimWidget classes

Create class SimWidget that will represent a rectangular widget for purposes of the layout simulation:

- A SimWidget should include instance variables minWidth, maxWidth, and prefHeight that store the layout constraints for this simulated widget
- The constructor for SimWidget should include incoming parameters for the three layout constraints
- A SimWidget should include instance variables myLeft, myTop, myWidth, and myHeight that store the current position and size of the simulated widget
- A SimWidget should create a Rectangle object that represents this widget's extents on screen (black border, yellow fill)
- Note that the SimWidget does not have any user interaction capabilities

Create class LinearLayout that will store child widgets:

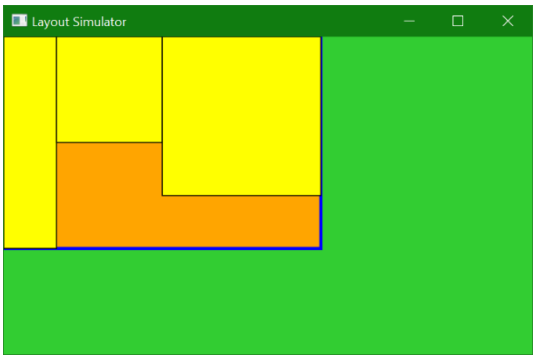
- LinearLayout should maintain a private list of children (i.e., the SimWidgets that will appear inside it) and allow children to be added to the list
- Write a method doLayout() that positions each child in the next available space from left to right, based on the child's minWidth
- A LinearLayout should create a Rectangle object that represents the layout container's extents on screen (purple border, orange fill)

Create class LayoutView that will set up, show, and control your layout simulator:

- LayoutView should extend Pane so that it can be added to your UI
- The constructor of LayoutView will set up the layout simulation. In this method:
    - Create an instance of LinearLayout named *root*
    - Create instances of class SimWidget and add them to root
    - Call root.doLayout() to do the initial layout
- Write method addRectangles(LinearLayout root) that will add the Rectangles of the LinearLayout and all of its children to the LayoutView

Example of a running system after part 1, with the following SimWidgets:

```
SimWidget sw1 = new SimWidget(50,200,200);
SimWidget sw2 = new SimWidget(100,100,100);
SimWidget sw3 = new SimWidget(150,250,150);
```



Part 2: Simple Layout

Add capabilities and methods to your classes to support the layout algorithm specified below and described in lectures.

For class SimWidget:

- Add method doHorizontalLayout(double parcelLeft, double parcelRight).
- Add an instance variable that allows specification of the SimWidget's vertical position: either TOP (at the top of its parcel), MIDDLE (the middle of the SimWidget is at the middle of its parcel), or FILL (the SimWidget is stretched from the top to the bottom of the parcel).
- Add method doVerticalLayout(double parcelTop, double parcelBottom) which will position the widget between the top and bottom values passed in to the method
- Layout rules:
    - All SimWidgets should fill their parcel horizontally
    - The height of the SimWidget should be the minimum of its prefHeight value and the height of the parcel

Add methods to class LinearLayout to lay out the children of a LinearLayout:

- calculateVIS() which will calculate the layout container's variable intrinsic size
- doHorizontalLayout(double parcelLeft, double parcelRight) which will lay out the container's children between the left and right values passed in to the method
- doVerticalLayout(double parcelTop, double parcelBottom) which will position the children between the top and bottom values passed in to the method
- Layout rules:
    - All children should be given their minWidth
    - If there is more horizontal space available, the space should be allocated equally – but children should never go beyond their maxWidth

Add code to class LayoutView to support the new layout capabilities:

- Attach listeners to the LayoutView for changes in the window size, as follows:

```
this.widthProperty().addListener((observable) -> root.doLayout(this.getWidth(),this.getHeight()));
this.heightProperty().addListener((observable) -> root.doLayout(this.getWidth(),this.getHeight()));
```

Example of a running system after part 2, with the following SimWidgets:

```
SimWidget sw1 = new SimWidget(50,200,200, SimWidget.VerticalPosition.MIDDLE);
SimWidget sw2 = new SimWidget(100,800,100, SimWidget.VerticalPosition.TOP);
SimWidget sw3 = new SimWidget(150,250,150, SimWidget.VerticalPosition.FILL);
```

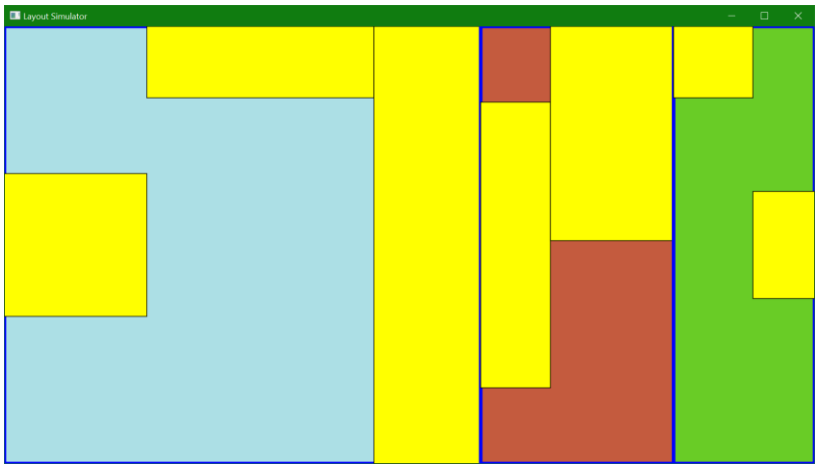
Part 3: Hierarchical Layout

Add code to your system to enable hierarchical layout – that is, the children of a LinearLayout can be either a SimWidget or a LinearLayout:

- Create a base class BaseWidget that will be the superclass of both SimWidget and LinearLayout
- Extract any common functionality from SimWidget and LinearLayout and place it in class BaseWidget
- The list of children should now be of type BaseWidget
- Add methods hasChildren() and getChildren() to class BaseWidget, to allow you to determine whether you are at a leaf node or an interior node in the containment hierarchy
- LinearLayouts should now be laid out like any other child in the horizontal dimension; in the vertical dimension, LinearLayouts always fill the parcel

Example of a running system after part 3, with the following setup code:

```
        root = new LinearLayout();
        SimWidget sw1 = new SimWidget(50,200,200, SimWidget.VerticalPosition.MIDDLE);
        root.addChild(sw1);
        SimWidget sw2 = new SimWidget(100,800,100, SimWidget.VerticalPosition.TOP);
        root.addChild(sw2);
        SimWidget sw3 = new SimWidget(50,150,150, SimWidget.VerticalPosition.FILL);
        root.addChild(sw3);
        LinearLayout hbox2 = new LinearLayout();
        SimWidget sw4 = new SimWidget(25,200,400, BaseWidget.VerticalPosition.MIDDLE);
        hbox2.addChild(sw4);
        SimWidget sw5 = new SimWidget(100,200,300, BaseWidget.VerticalPosition.TOP);
        hbox2.addChild(sw5);
        LinearLayout hbox3 = new LinearLayout();
        SimWidget sw6 = new SimWidget(75, 600, 100, BaseWidget.VerticalPosition.TOP);
        hbox3.addChild(sw6);
        SimWidget sw7 = new SimWidget(50,100,150, BaseWidget.VerticalPosition.MIDDLE);
        hbox3.addChild(sw7);
        hbox2.addChild(hbox3);
        root.addChild(hbox2);
```



What to hand in (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, and explains which elements of the assignment are working or not.
- Systems for 381 should never require the marker to install external libraries, other than JavaFX.
- This is an individual assignment – each student will hand in separately
- Review the material in the course syllabus regarding academic honesty, and follow all guidelines when completing this assignment. If you have any questions, contact your instructor

Where to hand in

Hand in your zip file to the Assignment 2 link on the course Canvas site.

Evaluation

- Part 1: all classes are implemented as specified, and the system produces graphical output that corresponds to the setup code that is in LayoutView (and the layout requirements specified in Part 1). Note that the markers will try different simulated layouts other than what is in your LayoutView class.

- Part 2: the main layout algorithm works as specified in Part 2, and the system produces graphical output that corresponds to the setup code that is in LayoutView, and that is responsive to changes in the size of the window based on the layout requirements specified in Part 2. All changes to classes are implemented as required. Note that the markers will try different simulated layouts other than what is in your LayoutView class.
- Part 3: hierarchical layout works as specified in Part 3, and is responsive to changes in the size of the window based on the layout requirements specified in Part 3. All changes to classes are implemented as required. Note that the markers will try different simulated layouts other than what is in your LayoutView class.
- Overall, your system should run correctly without errors, and your code should clearly demonstrate that you have satisfied the software requirements stated in the assignment description. (Document your code to assist the markers understand how you are satisfying the software requirements). Note that you may not use a JavaFX layout container (e.g., HBox) for your LinearLayout class instead of writing the layout code yourself.

If parts of your system have only partial implementations (e.g., a feature does not work but has been partially developed), clearly indicate this in your readme.txt file. Code should be appropriately documented and tested (although documentation will not be explicitly marked).

In general, no late assignments will be allowed, and no extensions will be given, except for emergency or medical reasons. (If you wish to use your one-time free extension, contact the instructors before the deadline at cmpt381@cs.usask.ca).