JavaFX Adventures part 3 - Design Considerations (+GridPane)

Due to the fact that the custom component should host a visual information much larger than it is possible to show on most displays, therefore it is just evident that a kind of scroll pane be used. The already existing Swing counterpart is a scroll pane, with a column header (that's the date and time information) and a row header (those are the resources). So first I have checked the FX ScrollPane (javafx.scene.control.ScrollPane). It turned out soon that the FX version has no headers - neither column, nor row header. There must be a reason why, but I don't really care. I decided at the very beginning that I'll never criticise the JavaFX - I'll just try to use it. That's just a fact that there are no headers of the FX ScrollPane, so I have to consider another solution.

> ⓘ **JavaFX limitation**
>
> The JavaFX ScrollPane has no column or row header, it maintains only the viewport (called "content"), and the two scroll bars (both are optional, and may appear only on demand). The FX ScrollPane is not a panel at all - it is a specialized control instead. Therefore, its getChildren() method - which is intended to add components to a parent - is not visible, and provides a dedicated method (setContent) to add the only available child - the viewport.

Creating a compound window and using it as the content of the scroll pane is not a real solution. Theoretically, I might be possible that upon scrolling, the content of this compound window be adjusted so that the header parts remain fixed (always shown). However, that doesn't really seem to be a reliable, bulletproof solution, so I had to consider other approaches. Soon I turned to the GridPane (javafx.scene.layout.GridPane). It lays out its children in a Grid, by using rows and columns. The width of the columns and the height of the rows might differ from each other. It is easy to achieve that all components within the same column be sized to have the same width, and within the same row to have the same height. In fact, that's the default behaviour. So I might have the following layout of this GridPane:

Header of the Row Header	The Column Header - that is, the Timeline view	Header of the Row Footer	Empty
The Row Header - that is, the resources the activities are assigned to.	The activities and the relations between the activities. The activities start at a specific point of time, end later, and they belong to resources.	The Row Footer - that is, the resources the activities are assigned to.	Main vertical scroll bar
Horitonzal scroll bar of the Row Header	Main horizontal scroll bar	Horizontal scroll bar of the Row Footer	Empty

Therefore, the resources might be either repeated, or being able to display different set of information on the right side - that's the "Row Footer". It is also shown that the resource view might be wider than the available space - hence the horizontal scroll bars for the row header and footer. This is possible, because the optional header of the row header or the header of the row footer are both supposed to have the same width than their corresponding resource view. However, the height of the timeline view must be fixed, and it cannot be scrolled - that's because the height of the header's and footer's header might have different height than the timeline view has.

There are rules to keep too. The horizontal scroll bar of the Row Header must control the scrolling of the Row Header, and the Header of the Row Header - those two must be synchronized when scrolled. Similarly, the horitontal scroll bar of the Row Footer must control the scrolling of the Row Footer, and the Header of the Row Footer - their horizontal scrolling is again synchronized. The main horizontal scroll bar must control the horizontal scrolling of the main viewport, and the time line simultaneously. The main vertical scroll bar must control the vertical scroll of the Row Header, the Main Viewport, and the Row Footer. Due to the fact that it is possible to scroll by using alternative input methods (mouse wheel, touchpad edges, etc.), therefore it must be ensured that whenever a view scrolls, the corresponding associated other views should be scrolled just the same.

Conclusion: the custom component is a grid pane, with six ScrollPanes and four ScrollBars added. The height of the first row should come from the Timeline view (it is accepted to introduce a limitation that no header might be any taller).The visible and the full width of the Left Resource View should come from that view (the visible width is the width of the first column). The visible and the full width of the Right Resource View should come from that view (and the visible width is the width of the third column). The visible and full width of the second column should come from the main viewport, which is the Activity View. The visible and full height of the second row should come from the Left Resource Pane. It will be a recommendation that both the Activity View and the Right Resource View have the same full height - the custom component might or might not enforce this rule, it could be decided later. The width and height of the scroll bars shouldn't be set explicitly, just let it be set automatically.

Controlling the rows and columns of a GridPane

```
    // *******************
    // Set the column constraints

    // leftWidth and all those are the preset widths coming from the corresponding models according to the
rules described above

    ColumnConstraints cc1 = new ColumnConstraints();
    cc1.setPrefWidth(leftWidth);
    cc1.setMaxWidth(leftWidth);
    cc1.setMinWidth(leftWidth);
    cc1.setHgrow(Priority.NEVER);

    ColumnConstraints cc3 = ColumnConstraintsBuilder
            .create()
            .hgrow(Priority.NEVER)
            .minWidth(rightWidth)
            .maxWidth(rightWidth)
            .prefWidth(rightWidth)
            .build();

        // Define the other column constraints similarly

    myCustomGripPane.getColumnConstraints().addAll(cc1, cc2, cc3, cc4);

    // *******************
    // Set the row constraints - it's very similar

    RowConstraints rc2 = new RowConstraints();
    rc2.setPrefHeight(height * visibleNumber);
    rc2.setMaxHeight(height * totalNumber);
    rc2.setVgrow(Priority.ALWAYS);

    // Define the other row constraints similarly

    myCustomGridPane.getRowConstraints().addAll(rc1, rc2, rc3);

    // *******************
    // Add the cells

    myCustomGridPane.add(rowheaderHeader, 0, 0);     // row, column
    myCustomGridPane.add(timelineView, 1, 0);
    // and so on
```

ⓘ **JavaFX option**

In most of the cases, JavaFX provides a builder for most of the controls and other objects. In the above code sample, both the traditional way of using setter methods, and the new way of using builders is presented when creating the column constraints. It's perhaps a personal taste which one is used, I cannot see any advantage or disadvantage of using one way or the other.

Now I have the general layout. The next question is, what should be the content of those six scroll panes. The constraints are that although the resources forming a kind of list, which might be represented as a table, but the timeline is not really a horizontally laid out table (unless I am prepared to use a table with hundreds of thousands of columns). That's because the resolution of the time line is one minute, but the total length might be one full year, thus I have 366*24*60 columns - that's 527,040 columns. That's not very practical to use a table like that. Also, considering that this general tool must be able to support logistics, and one resource may have even more activities on a daily basis, therefore one row might contain few hundred activities. Adding the fact that there might be several hundreds of resources, it doesn't seem to be adequate to use Controls as the representation of activities. So having panels as contents of the scroll panes, and adding controls to those panels is not the way to go. Having many (even hundreds of) thousands of controls on a view is definitely not the way to go - usually, the controls are resources of the underlying system. I do not have the urge to check this assumption currently. I will give it a try later; perhaps Java FX uses the controls differently than all those GUIs I met before.

So for now, the decision I made was to use Canvases, and draw upon those Canvases directly. Then the resources, the timeline, and the activities are all just painted (rendered) shapes like rectangles. Therefore, the adventure will continue with using the Canvas.

Previous pages:

Next pages:

part 4 - How to use Canvas