JavaFx: Border and Grid Pane Layout

Juedeja R. Richard

CSD: 402 Java for Programmers

Professor Josh Guardino

July 20, 2025

JavaFx: Border and Grid Pane Layout

The main purpose of JavaFx is to help developers create Graphical Interface Units (GUI) for their Java applications. When building an application, it is always helpful to create a layout pane for placing the GUI elements. Layout panes are containers used in JavaFx to place UI elements within the scene graph of a Java application. Layout panes are responsive- meaning when a window is resized, the panes automatically adjust to keep everything looking consistent and presentable. Items in a pane are called nodes. Nodes follow a tree data structure where parent classes like *Group, Region and Control* are considered branch nodes while leaf nodes are classes like *Rectangle, Text, ImageView and MediaView*. Leaf nodes cannot have children and only one node, the root node, does not have a parent class. Java provides many layout panes for developers to work with, two of which are Border and Grid Pane.

Border Pane places content in 5 regions: left, right, top, bottom and center. They can be adjusted to any size you need, and if you don't want to use all the regions, you can simply leave them undefined, and no space will be left for them. Border Pane is considered a classic layout that you see with most websites- a navbar on top, the footer on the bottom, navigation on the left, extra information on the right and the main content in the center. It can be accessed with the *java.scene.layout.borderpane* class and provides methods like *setTop()* and *setBottom()* or *setPreferredHeight()*. It is important to note that changing the height or width of any of the regions will affect the others. The top and bottom regions are usually set to their preferred heights, which can alter the heights of the left, right, and center regions. The left and right regions are set to their preferred widths, which then alter the widths of the top and bottom regions. Content is not clipped automatically when it extends outside of the child regions range

JavaFx: Border and Grid Pane Layout

so it could overlap content in another region. The Border pane constructor is just *Borderpane()* to create the layout. Inside the parentheses of the constructor, you can list what nodes are in what regions. For example, *Borderpane(Node left)* will be used to represent the left region of the border pane layout.

Grid Pane layout places its content in rows and columns. These nodes can be placed anywhere in the grid and can also span multiple rows and columns. Nodes are numbered from 0 to n and can overlap their content in that order. Basically, 0 would be the content in the background and n would be the content in the foreground. Any number in between will be sandwiched in the middle of those two. When parameters are not set, the child node will be placed in the first row or column and the number of columns/rows will be 1 by default. A Grid Pane layout uses coordinates like those on a map to locate where a child node is on the grid. For example, the very first area in grid layout with 3 columns and 2 rows in the top left would have a location of (0,0). The area to the right of it in the second column would (1,0), the third column would be (2,0) and so on. Some common methods are *add(),* which is used to place a component at a specific coordinate, *setVgap()* which defines the amount of vertical space between components, and *setHgap()* which does the same thing for horizontal space. You can also set the size of rows and columns by using *ColumnConstraints, RowConstraints or setPercentWidth*. A row or column can be given priority to take available extra space by setting its grow property to *ALWAYS.*

Essentially, both Border Pane and Grid Pane are the basis for creating decent layouts for JavaFx applications. For standard layouts that everyone recognizes, Border Pane is the way to

go. For something a bit more unique and complex, Grid Pane gives you more freedom and flexibility. Which one you use is entirely up to you and the needs of your project.

References

BorderPane (JavaFX 21). (2023). Openjfx.io. https://openjfx.io/javadoc/21/javafx.graphics/javafx/scene/layout/BorderPane.html

GridPane (JavaFX 8). (n.d.). Docs.oracle.com. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/GridPane.html

JavaFX BorderPane Layout. (2025). Tutorialspoint.com; TutorialsPoint. https://www.tutorialspoint.com/javafx/layout_borderpane.htm

Ramesh Fadatare. (2020, October 10). *JavaFx BorderPane Example*. Javaguides.net; Blogger. https://www.javaguides.net/2020/10/javafx-borderpane-example-tutorial.html

JavaFX GridPane Layout | Java GUI - CodersLegacy. (2020, August 9). CodersLegacy. https://coderslegacy.com/java/javafx-gridpane-layout/

Bodnar, J. (2023). *JavaFX layout panes*. Zetcode.com. https://zetcode.com/gui/javafx/layoutpanes/

Node (JavaFX 8). (2015, February 10). Oracle.com. https://docs.oracle.com/javase/8//javafx/api/javafx/scene/Node.html

Working With Layouts in JavaFX: Using Built-in Layout Panes | JavaFX 2 Tutorials and Documentation. (2013). Oracle.com. https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm