JavaFX Scenes, Controls and Layouts

UI elements in JavaFX applications are commonly referred to as **controls**. Some common controls include:

**Label**

A label control displays text.

import javafx.scene.control.Label;

**TextField**

A TextField control is a rectangular field that can accept keyboard input from the user.

import javafx.scene.control.TextField;

**Button**

A Button is a rectangular control that appears as a button with a caption written across its face. Clicking the button triggers an action.

import javafx.scene.control.Button;

# Creating a JavaFX Scene

The general steps to follow when creating a JavaFX scene are:

1. Create the controls that will be in the scene.
2. Create a layout container of some type, and add the controls to the layout container.
3. Create a `Scene` object, and add the layout container to the `Scene` object.
4. Add the `Scene` object to the stage.

The constructor of the `Scene` class has the following overloads:

public Scene(Parent root)

public Scene(Parent root, double width, double height)

public Scene(Parent root, double width, double height, Paint fill)

public Scene(Parent root, Paint fill)

The `Parent` class is a superclass for all nodes (in a scene graph) that can have children. This includes both layouts and controls (only ones that can have children).

The `root` parameter takes the root node of the scene graph as an argument (therefore an empty scene with no root node cannot be created).

The `width` and `height` parameters take the width and height of the scene. You may be wondering why they are of the type `double`, when a monitor cannot divide pixels into fractional parts (i.e., render half a pixel as one colour and the other half as another). There are a few reasons for this:

* Modern rendering engines often perform calculations in floating-point to achieve higher precision, even if the final output is rounded to integer pixel values.
* During animations or transitions, intermediate values are often fractional.
* High-DPI displays scale content to appear consistent across different pixel densities. For instance, a UI element specified as 200.5 logical pixels may map to a higher number of physical pixels on a Retina display. Using a double for dimensions accommodates such scaling more gracefully.
* When applying transformations (like rotations or scaling) to nodes within a scene, the calculations often result in fractional positions and sizes. Having dimensions as doubles ensures that these transformations are calculated with maximum precision.

***\*When you create a scene without specifying a size, it will default to the minimum possible size that can display all its contents, including any spacing or padding.***

***When you create a layout (such as an HBox or VBox) without specifying its size, it will default to the minimum possible size that can display all its contents, including any spacing or padding (the exception is for root node layouts, which will stretch to fit their `Scene`).***

***When you create a control (such as a Label or Button) without specifying its size, its size will be determined by its intrinsic properties and the content the control contains. For example, the size of a Label is determined by its content (it will default to the smallest size that accommodates all its content including padding and spacing). The situation is similar for a Button, however, a Button also has a default minimum size which it will be if no content is added to it.***

## Aligning Controls in an HBox Layout Container

Elements within a JavaFX layout container are arranged according to the layout's alignment. Each type of layout has a default alignment that will be used if an alignment is not explicitly specified.

The default alignment of an HBox container is `TOP\_LEFT`. This means that controls will be aligned from left to right. The height of the HBox will be equal to the height of the control with the largest height. Any smaller controls will be aligned along the top of the HBox with empty space below them.

If the alignment were set to `BOTTOM\_LEFT`, the smaller controls would be aligned along the bottom of the HBox with empty space above them.

|  |  |
| --- | --- |
| HBox with `TOP\_LEFT` alignment.  A screenshot of a computer  Description automatically generated | HBox with `BOTTOM\_LEFT` alignment.  A screenshot of a computer  Description automatically generated |

### `Pos` Data Type

In JavaFX, `Pos` is an enum type defined in the `javafx.geometry` package. It is used to represent the alignment positions within layout panes and other nodes.

The `HBox` class, as well as many other layout types, have a `.setAlignment(Pos value)` method which takes an enum constant as an argument. These are listed below:

|  |  |  |
| --- | --- | --- |
| Pos.TOP\_LEFT | Pos.TOP\_CENTER | Pos.TOP\_RIGHT |
| Pos.CENTER\_LEFT | Pos.CENTER | Pos.CENTER\_RIGHT |
| Pos.BOTTOM\_LEFT | Pos.BOTTOM\_CENTER | Pos.BOTTOM\_RIGHT |

To set the alignment of a layout container, you will first need to import the `Pos` class using the following statement:

import javafx.geometry.Pos;

Then, you can pass a Pos constant as an argument to the following method:

hbox.setAlignment(Pos.CENTER);

**Root Node Alignment and Sizing**

The root node of a `Scene` will stretch to fill the entire scene. Therefore, it does not technically have any alignment within the Scene.