Layout Panes

Following are the various Layout panes (classes) provided by JavaFX. These classes exist in the package **javafx.scene.layout.**

S.No	Shape & Description
1	**HBox** : The HBox layout arranges all the nodes in our application in a single horizontal row.
2	**VBox** : The VBox layout arranges all the nodes in our application in a single vertical column.
3	**BorderPane** : The Border Pane layout arranges the nodes in our application in top, left, right, bottom and center positions.
4	**StackPane** : The stack pane layout arranges the nodes in our application on top of another just like in a stack. The node added first is placed at the bottom of the stack and the next node is placed on top of it.
5	**TextFlow** : The Text Flow layout arranges multiple text nodes in a single flow.
6	**AnchorPane** : The Anchor pane layout anchors the nodes in our application at a particular distance from the pane.
7	**TilePane** : The Tile Pane layout adds all the nodes of our application in the form of uniformly sized tiles.
8	**GridPane** : The Grid Pane layout arranges the nodes in our application as a grid of rows and columns. This layout comes handy while creating forms using JavaFX.
9	**FlowPane** : The flow pane layout wraps all the nodes in a flow. A horizontal flow pane wraps the elements of the pane at its height, while a vertical flow pane wraps the elements at its width.

HBox

If we use HBox in the layout in our application, all the nodes are set in a single horizontal row.
The class named **HBox** of the package **javafx.scene.layout** represents the HBox pane. This class contains five properties namely −
alignment − This property represents the alignment of the nodes in the bounds of the HBox. You can set value to this property using the setter method **setAlignment()**.
fillHeight − This property is of Boolean type and on setting this to true, the resizable nodes in the HBox are resized to the height of the HBox. You can set value to this property using the setter method **setFillHeight()**.
spacing − This property is of double type and it represents the space between the children of the HBox. You can set value to this property using the setter method **setSpacing()**.
In addition to these, this class also provides a couple of methods, which are −
setHgrow() − Sets the horizontal grow priority for the child when contained by an HBox. This method accepts a node and a priority value.

setMargin() – Using this method, you can set margins to the HBox. This method accepts a node and an object of the Insets class (A set of inside offsets for the 4 side of a rectangular area).

Example: The following program is an example of the HBox layout. Here, we are inserting a text field and two buttons, play and stop. This is done with a spacing of 10 and each having margins with dimensions – (10, 10, 10, 10).

```
package javafxapplication6;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.TextField;

public class JavaFXApplication6 extends Application {

    @Override
    public void start(Stage stage) {
      //creating a text field
      TextField textField = new TextField();

      //Creating the play button
      Button playButton = new Button("Play");

      //Creating the stop button
       Button stopButton = new Button("stop");

      //Instantiating the HBox class
      HBox hbox = new HBox();

      //Setting the space between the nodes of a HBox pane
      hbox.setSpacing(10);

      //Setting the margin to the nodes
      hbox.setMargin(textField, new Insets(20, 20, 20, 20));
      hbox.setMargin(playButton, new Insets(20, 20, 20, 20));
      hbox.setMargin(stopButton, new Insets(20, 20, 20, 60));

      //retrieving the observable list of the HBox
```

```
        ObservableList list = hbox.getChildren();

        //Adding all the nodes to the observable list (HBox)
        list.addAll(textField, playButton, stopButton);

        //Creating a scene object
        Scene scene = new Scene(hbox);

        //Setting title to the Stage
        stage.setTitle("Hbox Example");

        //Adding scene to the stage
        stage.setScene(scene);

        //Displaying the contents of the stage
        stage.show();
      }
      public static void main(String args[]){
        launch(args);
      }
}
```

On executing, the above program generates a JavaFX window as shown below.

VBox

If we use VBox as the layout in our application, all the nodes are set in a single vertical column.

The class named **VBox** of the package **javafx.scene.layout** represents the VBox pane. This class contains five properties, which are −

alignment − This property represents the alignment of the nodes inside the bounds of the VBox. You can set value to this property by using the setter method **setAlignment()**.

fillHeight − This property is of Boolean type and on setting this to be true; the resizable nodes in the VBox are resized to the height of the VBox. You can set value to this property using the setter method **setFillHeight()**.

spacing − This property is of double type and it represents the space between the children of the VBox. You can set value to this property using the setter method **setSpacing()**.

In addition to these, this class also provides the following methods −

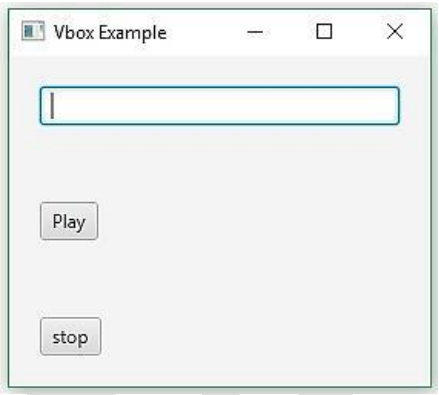
setVgrow() − Sets the vertical grow priority for the child when contained by a VBox. This method accepts a node and a priority value.

setMargin() − Using this method, you can set margins to the VBox. This method accepts a node and an object of the Insets class (A set of inside offsets for the 4 sides of a rectangular area)

Example: The following program is an example of the **VBox** layout. In this, we are inserting a text field and two buttons, play and stop. This is done with a spacing of 10 and each having margins with dimensions – (10, 10, 10, 10).

```
package javafxapplication7;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.scene.control.Button;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.TextField;

public class JavaFXApplication7 extends Application {

    @Override
    public void start(Stage stage) {
      //creating a text field
      TextField textField = new TextField();

      //Creating the play button
      Button playButton = new Button("Play");

      //Creating the stop button
      Button stopButton = new Button("stop");

      //Instantiating the VBox class
      VBox vBox = new VBox();

      //Setting the space between the nodes of a VBox pane
      vBox.setSpacing(10);

      //Setting the margin to the nodes
      vBox.setMargin(textField, new Insets(20, 20, 20, 20));
      vBox.setMargin(playButton, new Insets(20, 20, 20, 20));
      vBox.setMargin(stopButton, new Insets(20, 20, 20, 20));

      //retrieving the observable list of the VBox
      ObservableList list = vBox.getChildren();

      //Adding all the nodes to the observable list
      list.addAll(textField, playButton, stopButton);
```

```
        //Creating a scene object
        Scene scene = new Scene(vBox);

        //Setting title to the Stage
        stage.setTitle("Vbox Example");

        //Adding scene to the stage
        stage.setScene(scene);

        //Displaying the contents of the stage
        stage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```



On executing, the above program generates a JavaFX window as shown below.