Application Program Development

Segment : Automatic Resizing using Layouts

Mahboob Ali

Outcomes

- Understanding more about layouts
- Understanding of Menu in JavaFX
- Creating our own dialog boxes
- JavaFX beans, properties and its binding with the components

Automatic resizing using layouts

- JAVA FX provides a mechanism called a ***Layout Pane*** that allows the automatic arrangement (i.e., "laying out") of the components of an application as the window is resized.
- Why should we use a layout pane ?
  - we would not have to compute locations and sizes for our components
  - our components will resize automatically when the window is resized
  - our interface will appear "nicely" on all platforms

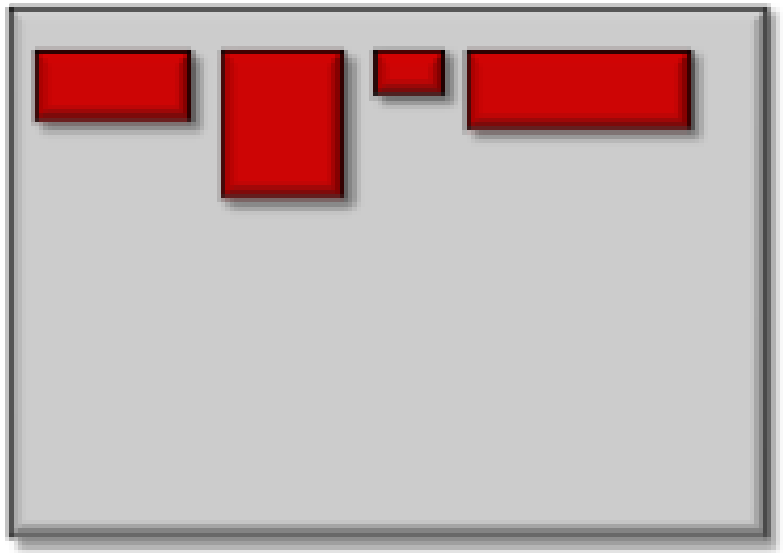
- In JAVA FX, each layout defines methods necessary for a class to be able to arrange **Components** within a **Container.**
- There are some commonly used layout pane classes that we can use.
- We will discuss the following, although there are more available:

  FlowPane, BorderPane, **HBox**, **VBox**, and **GridPane**

Note: *Layout panes are "set" for a pane using the* ***setLayout()*** *method. If set to* ***null****, then no layout manager is used.*

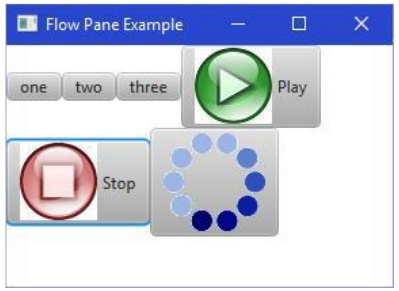
FlowPane

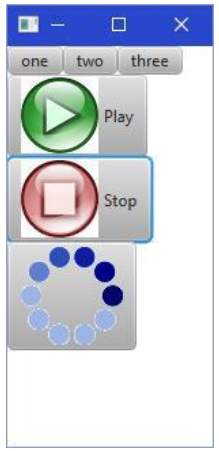
- The simplest layout pane is the **FlowPane**.
- It is commonly used to arrange just a few components on a pane.
- With this pane, components (e.g., buttons, text fields, etc..) are arranged horizontally from left to right ... like lines of words in a paragraph written in English.
- If no space remains on the current line, components flow (or wrap around) to the next "line".
- The height of each line is the maximum height of any component on that line.
- By default, components are centered horizontally on each line, but this can be changed.



```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;
import javafx.scene.image.*;

public class FlowPaneExample extends Application {
        public void start(Stage primaryStage) {
                FlowPane aPane = new FlowPane();
                aPane.getChildren().add(new Button("one"));
                aPane.getChildren().add(new Button("two"));
                aPane.getChildren().add(new Button("three"));
                aPane.getChildren().add(new Button("Play", new ImageView(
                      new Image(getClass().getResourceAsStream("GreenButton.jpg")))));
                aPane.getChildren().add(new Button("Stop", new ImageView(
                        new Image(getClass().getResourceAsStream("RedButton.jpg")))));
                Button b = new Button();
                b.setGraphic(new ImageView( new
                       Image(getClass().getResourceAsStream("Progress.gif"))));
                aPane.getChildren().add(b);
                primaryStage.setTitle("Flow Pane Example");
                primaryStage.setScene(new Scene(aPane, 500,100));
                primaryStage.show();
        }
        public static void main(String[] args) { launch(args); }
}
```

Results of the previous slide code with different size's





- We can also specify spacing between components as well as spacing around the pane's border.
- For example, we can use **setVgap()** to specify the vertical gap that we want to leave between each row of components as the components wrap around


```
aPane.setVgap(0);
```


```
aPane.setVgap(10);
```


```
aPane.setVgap(30);
```

Results of the previous slide code with different size's

- We can also use **setHgap()** to specify the horizontal gap that we want to leave between each column of components


```
aPane.setHgap(0);
```


```
aPane.setHgap(30);
```

- Lastly, we can specify the margins around the border of the frame by using **setPadding()** as follows:

  aPane.setPadding(new Insets(20, 10, 30, 40));

- This will set the margin to be 20 pixels on the top, 10 pixels on the right, 30 pixels on the bottom and 40 pixels on the left.
- However, since our pane is by itself in the window, only the top and left settings make sense in this application.
- In the case where we want the same margin on the top, right, bottom and left sides, we can use this simpler constructor instead:

  ```
  aPane.setPadding(new Insets(25));
  ```

  which will set the margin to 25 on all 4 sides


```
aPane.setPadding(new Insets(25,0,0,25));
```

HBox/ VBox

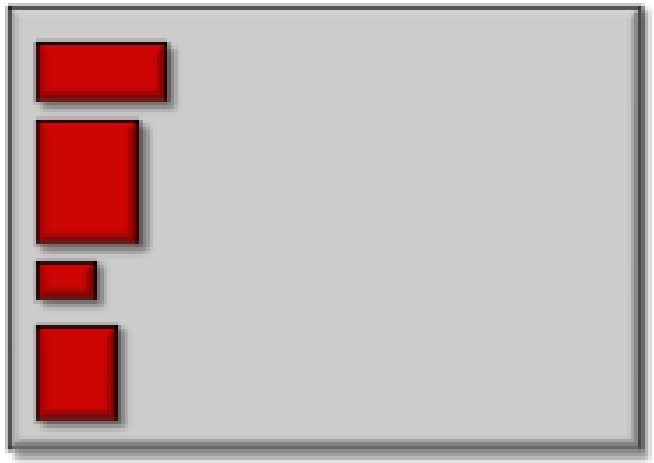
- The **HBox** and **VBox** layouts are also very simple to use.
- It is similar to the **FlowPane** in that it arranges components one after another, either horizontally or vertically.
- It does not have a wrap-around effect.
- Any components that do not fit on the line are simply not shown.
- If we want to lay the components out horizontally, we use

```
new HBox()
```

as our pane.

- To lay the components out vertically, we use

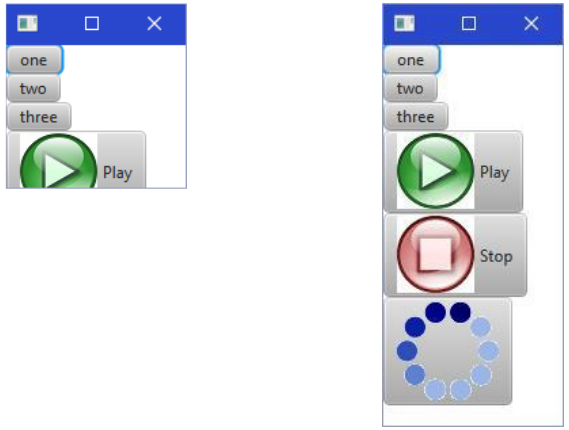
```
new VBox()
```



```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import javafx.scene.image.*;

public class HBoxExample extends Application {
	public void start(Stage primaryStage) {
		HBox aPane = new HBox();
		aPane.getChildren().add(new Button("one"));
		aPane.getChildren().add(new Button("two"));
		aPane.getChildren().add(new Button("three"));
		aPane.getChildren().add(new Button("Play", new ImageView(
				new Image(getClass()
					.getResourceAsStream("GreenButton.jpg")))));
		aPane.getChildren().add(new Button("Stop", new ImageView(
				new Image(getClass()
					.getResourceAsStream("RedButton.jpg")))));
		Button b = new Button();
		b.setGraphic(new ImageView( new Image(getClass()
					.getResourceAsStream("Progress.gif"))));
		aPane.getChildren().add(b);
		primaryStage.setTitle("HBox Example");
		primaryStage.setScene(new Scene(aPane, 500,100));
		primaryStage.show();
	}
	public static void main(String[] args) { launch(args); }
}
```

Results of the previous slide code with different size's

HBox result



VBox result

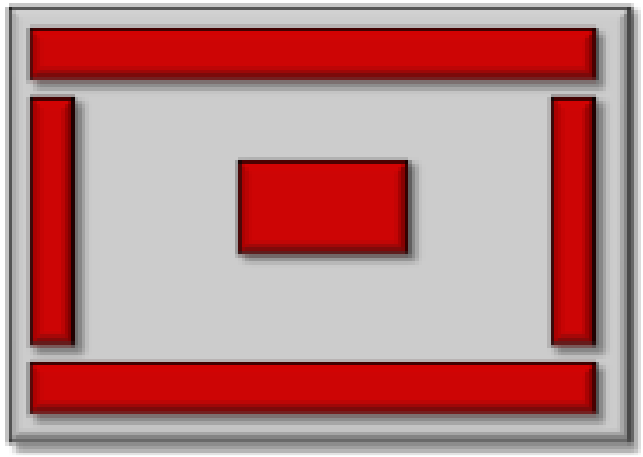
- As with the **FlowPane**, we can specify the **Insets** as well as spacing between components


```
aPane.setPadding(new Insets(10));
aPane.setSpacing(5);
```

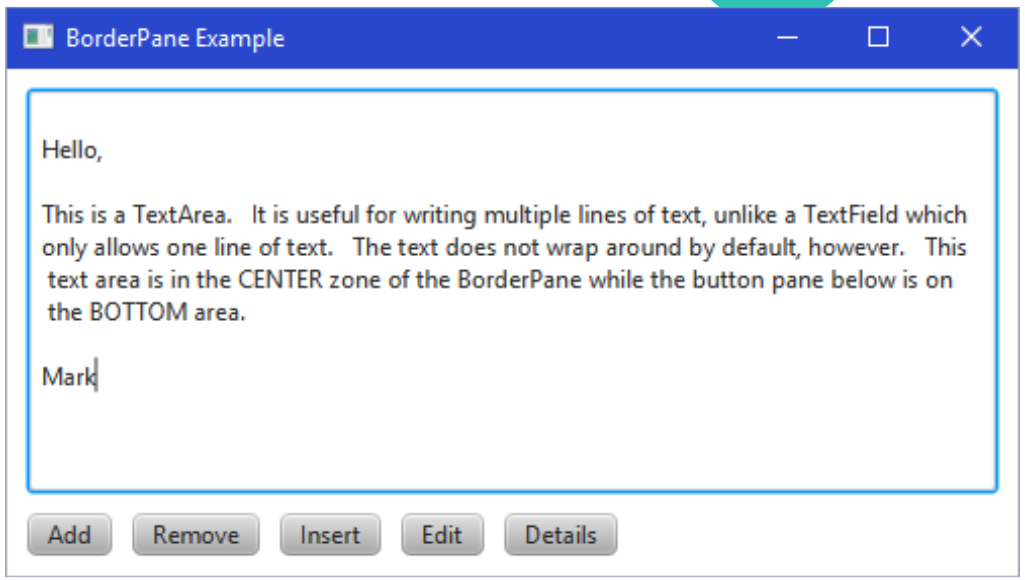
BorderPane

- The **BorderPane** is a very useful layout.
- Instead of re-arranging components, it allows you to place components at one of five anchored positions on the window (i.e., **top**, **left**, **bottom**, **right** or **center**).
- As the window resizes, components stay *"anchored"* to the side of the window or to its center.
- The components will grow accordingly.
- You may place at most one component in each of the 5 anchored positions ... but this one component may be a container such as another **Pane** that contains other components inside of it.
- Typically, you do NOT place a component in each of the 5 areas but choose just a few of the areas.
- We can add a componentOrPane to one of 5 areas of a **BorderPane** by using one of the following methods:

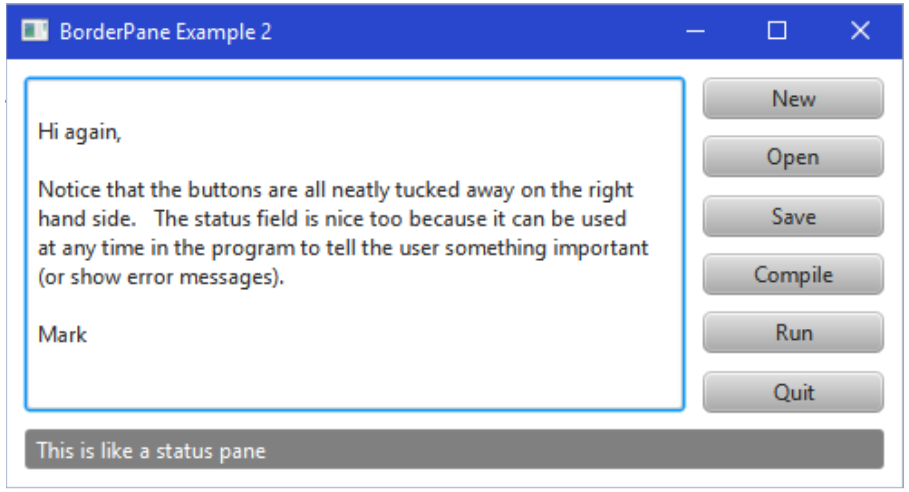
```
aBorderPane = new BorderPane();
componentOrPane1 = ...;
componentOrPane2 = ...;
...
aBorderPane.setTop(componentOrPane1);
aBorderPane.setRight(componentOrPane2);
aBorderPane.setBottom(componentOrPane3);
aBorderPane.setLeft(componentOrPane4);
aBorderPane.setCenter(componentOrPane5);
```



```java
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;

public class BorderPaneExample extends Application {
        public void start(Stage primaryStage) {
                BorderPane aPane = new BorderPane();
                aPane.setPadding(new Insets(10));
                FlowPane buttonPane = new FlowPane();
                buttonPane.setPadding(new Insets(10, 0, 0, 0));
                buttonPane.setHgap(10);
                buttonPane.getChildren().add(new Button("Add"));
                buttonPane.getChildren().add(new Button("Remove"));
                buttonPane.getChildren().add(new Button("Insert"));
                buttonPane.getChildren().add(new Button("Edit"));
                buttonPane.getChildren().add(new Button("Details"));
                aPane.setBottom(buttonPane); aPane.setCenter(new TextArea());
                primaryStage.setTitle("BorderPane Example");
                primaryStage.setScene(new Scene(aPane, 500,250));
                primaryStage.show();
        }
        public static void main(String[] args) { launch(args); }
}
```



```java
import javafx.application.Application; import javafx.geometry.Insets;
import javafx.scene.Scene; import javafx.scene.control.*;
import javafx.scene.layout.*; import javafx.stage.Stage;

public class BorderPaneExample2 extends Application {
        public void start(Stage primaryStage) {
                Button[] buttons; String[] names = {"New", "Open", "Save", "Compile", "Run", "Quit"};
                 BorderPane aPane = new BorderPane();
                 aPane.setPadding(new Insets(10));
                 VBox buttonPane = new VBox();
                 buttonPane.setPadding(new Insets(0, 0, 0, 10));
                 buttonPane.setSpacing(10); buttons = new Button[names.length];
                 for (int i=0; i<names.length; i++) {
                         buttons[i] = new Button(names[i]);
                         buttons[i].setPrefWidth(100); buttons[i].setPrefHeight(30);
                         buttonPane.getChildren().add(buttons[i]);        }
                 aPane.setRight(buttonPane); aPane.setCenter(new TextArea());
                 TextField statusField = new TextField("This is like a status pane");
                 statusField.setStyle("-fx-background-color: GRAY; -fx-text-fill: WHITE;");
                 aPane.setMargin(statusField, new Insets(10,0,0,0)); // allows spacing at top
                 aPane.setBottom(statusField);
                 primaryStage.setTitle("BorderPane Example 2");
                 primaryStage.setScene(new Scene(aPane, 500,500))
                 primaryStage.show();
        }
        public static void main(String[] args) { launch(args); }
}
```

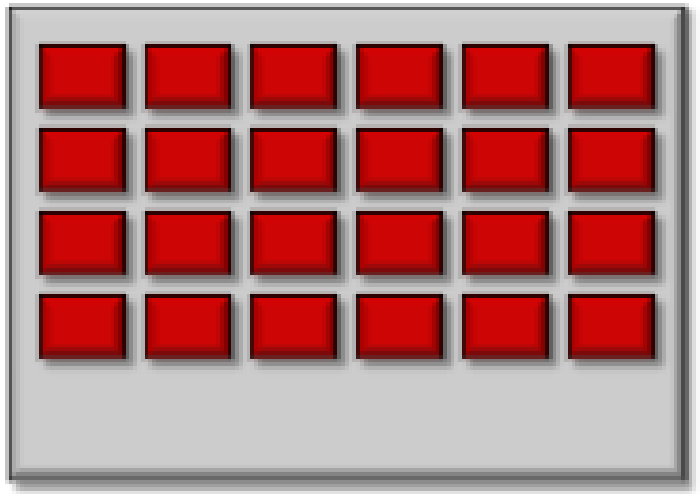


Simple GridPane

- A GridPane is excellent for arranging a 2-dimensional grid of components (such as buttons on a keypad).
- It automatically aligns the components neatly into rows and columns.
- The components are all of the same size, however you can add different sized components as well.
- Components are added by specifying their column and row in the grid.

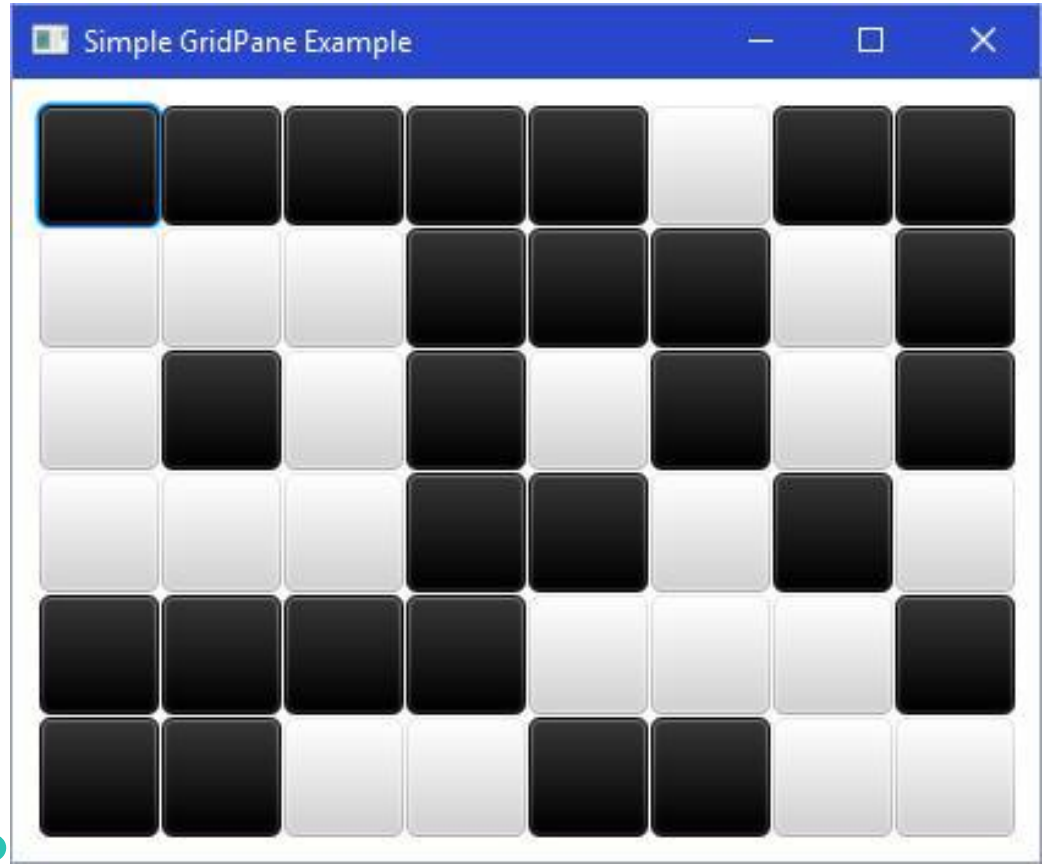
```
aGridPane.add(aComponent, col, row);
```

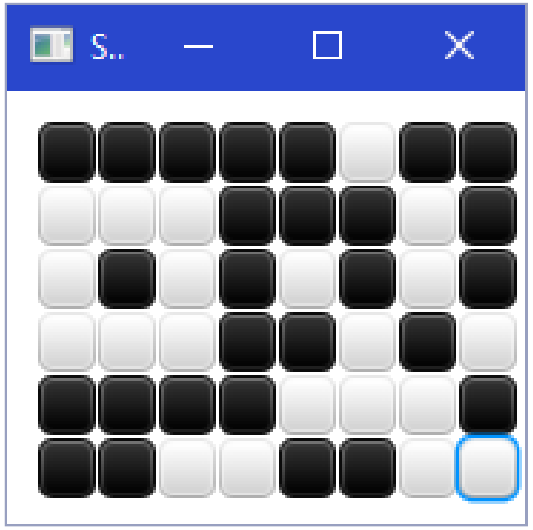
- JAVA determines the number of rows and columns to use for the grid by considering all of the **row** and **col** parameters that you use in these **add()** method calls.
- **setHgap()** and **setVgap()** specify the horizontal and vertical margin (in pixels) between components and the **setPadding()** allows you to specify margins around the outside of the pane.



```java
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene; import javafx.scene.control.Button;
import javafx.scene.layout.GridPane; import javafx.stage.Stage;

public class GridPaneExample extends Application {
	public void start(Stage primaryStage) {
		GridPane aPane = new GridPane();
		aPane.setPadding(new Insets(10, 10, 10, 10));
		aPane.setHgap(1);
		aPane.setVgap(1);
		for (int row=1; row<=6; row++)
			for (int col=1; col<=8; col++) {
				Button b = new Button();
				// Make the buttons bigger than we want. They will be
				// re-sized to fit within the shrunken pane.
				b.setPrefWidth(200);
				b.setPrefHeight(200);
				if
					(Math.random() < 0.5) b.setStyle("-fx-base: WHITE;");
				else
					b.setStyle("-fx-base: BLACK;");
				aPane.add(b, col, row);
			}
		primaryStage.setTitle("Simple GridPane Example");
		primaryStage.setScene(new Scene(aPane, 420,320));
		primaryStage.show();
	}
	public static void main(String[] args) { launch(args); }
}
```

Results of the previous slide code with different size's

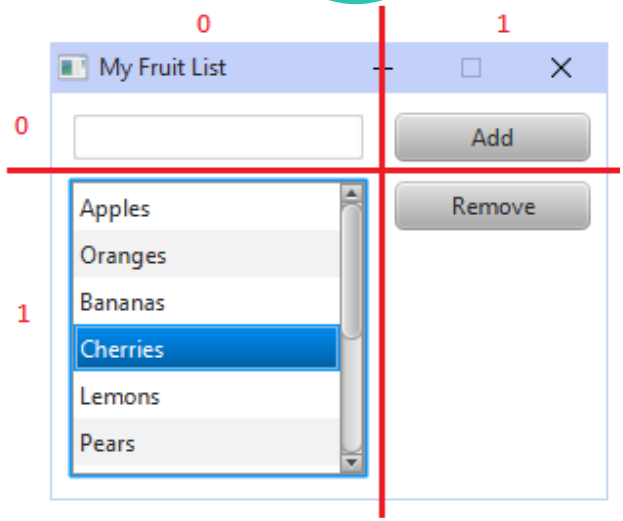




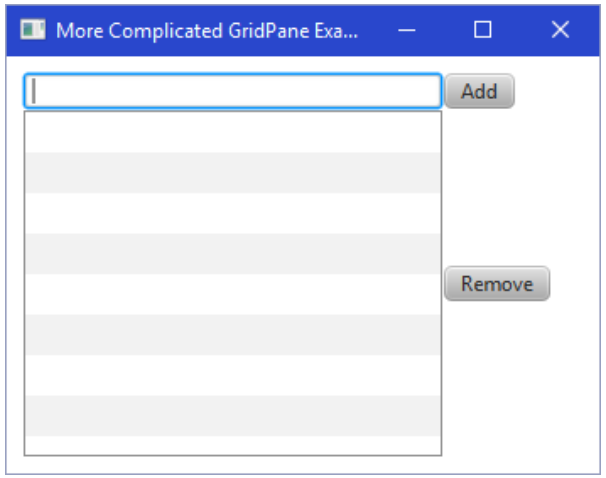
More Complicated GridPane

- The **GridPane** can also be the most flexible of all the layout panes.
- It allows you to be very specific in the placement of all components and to indicate exactly how each component is to resize as the window shrinks or grows.
- However, due to the flexibility of this layout, it is more complicated to use than any of the other layouts.
- The **GridPane** can also arrange components in a non-uniform grid where the grid rows and columns are not explicitly defined.
- It may be non-uniform in that the rows and columns may have variable heights and widths.
- Also, each component can occupy (i.e., span) multiple rows and columns.
- Let us see couple of examples showing how we can follow some simple steps to create a window with nicely arranged components that resizes in a nice, consistent manner.

- How can we use a **GridPane** so that the window resizes in a way that re-arranges the components nicely ?
  - Well, we can start by determining the components that lie in the same row and column.
  - To do this, we just need to "imagine" some lines between components both vertically and horizontally as shown here.
  - This forms a grid. We then number the columns and rows, starting at 0 at the top left.
  - This will be the basis for laying out the components.
  - We can lay out all the components by specifying the grid location (i.e., column and row) that each component lies in when we add it to the pane




```
GridPane aPane = new GridPane();

TextField newItemField = new TextField();
aPane.add(newItemField,0,0);

Button addButton = new Button("Add");

aPane.add(addButton,1,0);

ListView<String> fruitList = new ListView<String>();

aPane.add(fruitList,0,1);

Button removeButton = new Button("Remove");

aPane.add(removeButton,1,1);
```

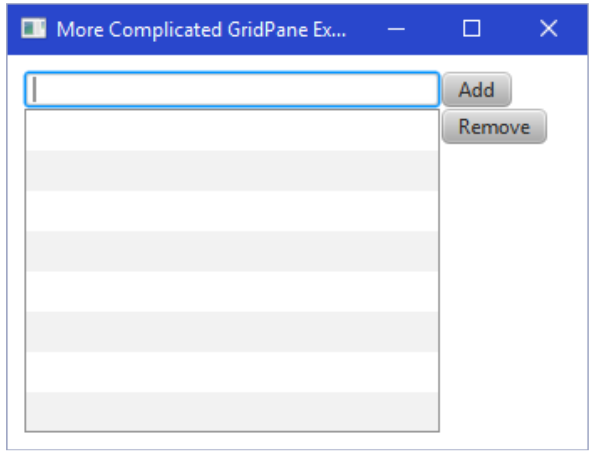


- we need to move the **Remove** button up, make the buttons the same size and add some spacing.
- Currently, the Remove button is centered vertically. We can change this by using:

```
aPane.setValignment(removeButton, VPos.TOP);
```

- The **setValignment()** allows us to set the alignment of a component to `VPos.TOP`, `VPos.BOTTOM`, or `VPos.CENTER`.

Here is the result ---------->

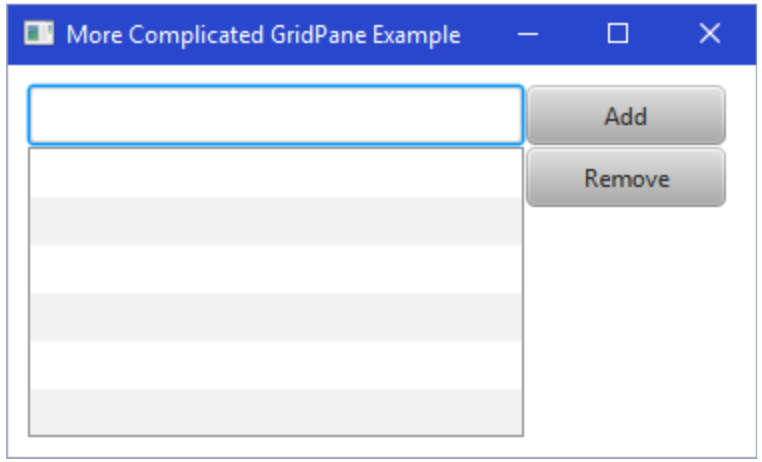


- Now, we need to make the buttons the same size.
- The simplest way to do this is to specify the width and height that we want the buttons to have.
- We can make the buttons 100x30 in size by setting the minimum width and height as follows:

```
addButton.setMinHeight(30); addButton.setMinWidth(100);
removeButton.setMinHeight(30);
removeButton.setMinWidth(100);
```

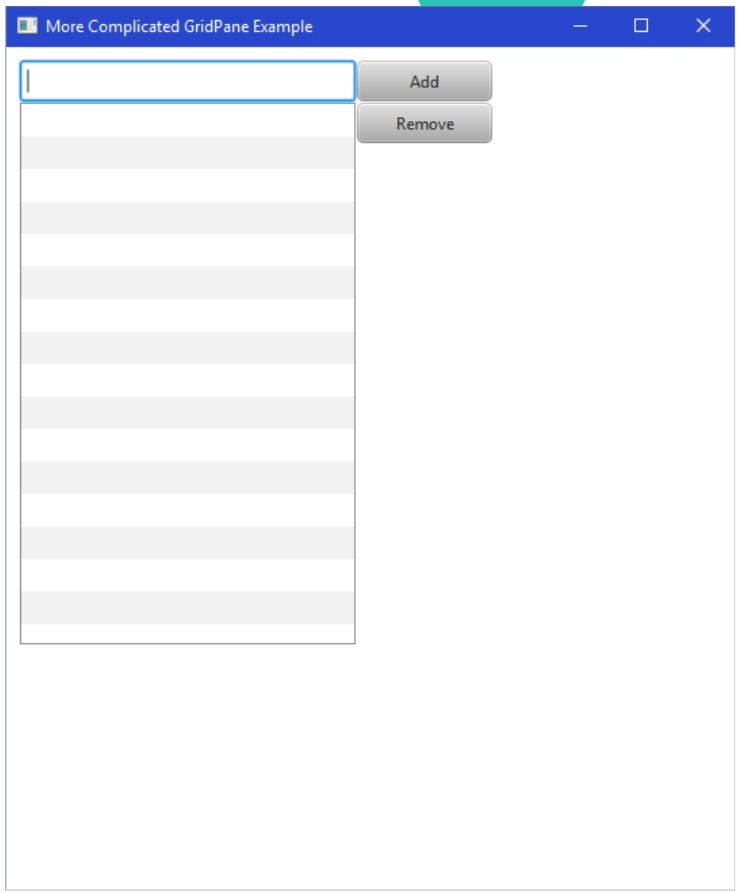
- We will also set the TextField to have the same height as the buttons:

```
newItemField.setMinHeight(30);
```



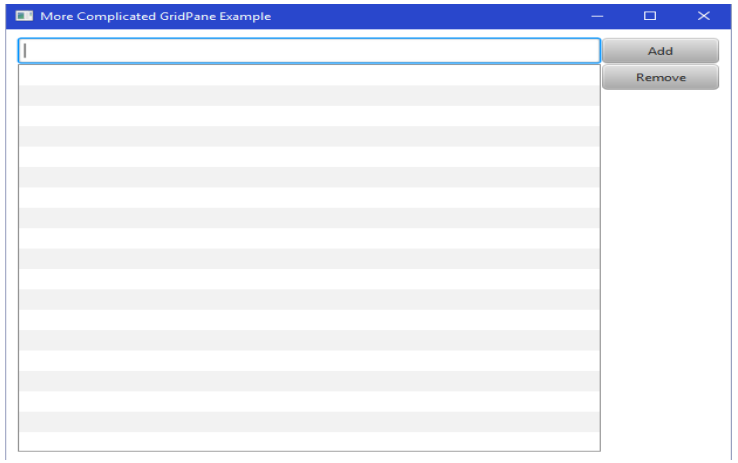
- The window does not resize properly as shown here on the right.
- When the window is enlarged, the components do not grow properly to take up the extra window space.
- We fix this by simply specifying that we want the fruitList to be as large as possible to take up all that extra space.
- To do this, we simply set the preferred width and height to the largest possible values:

```
fruitList.setPrefWidth(Integer.MAX_VALUE);
fruitList.setPrefHeight(Integer.MAX_VALUE);
```



- Now the list resizes nicely when the window grows as shown here on the right ---------->
- The last thing to do is to adjust the spacing around the components. We can use the **setMargin()** method for our components as follows:

```
aPane.setMargin(newItemField, new Insets(0, 0, 10, 0));
aPane.setMargin(addButton, new Insets(0, 0, 10, 10));
aPane.setMargin(removeButton, new Insets(0, 0, 0, 10));
```

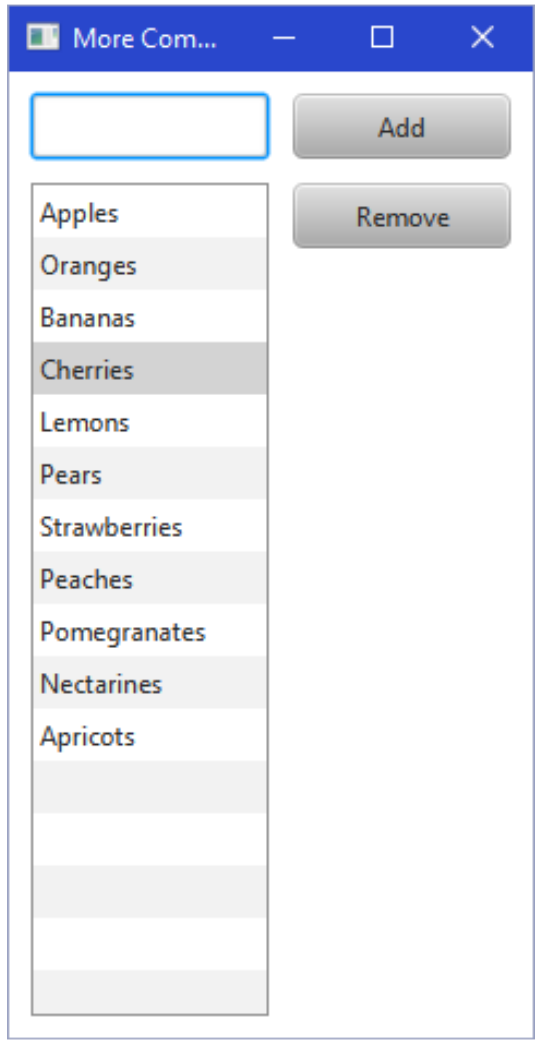


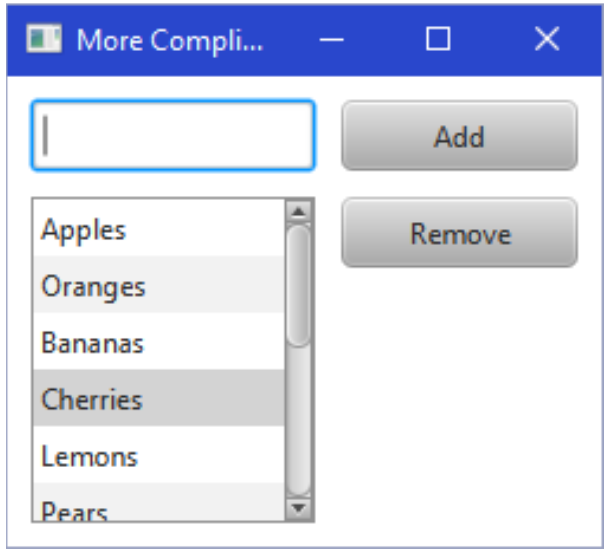
Complete Code

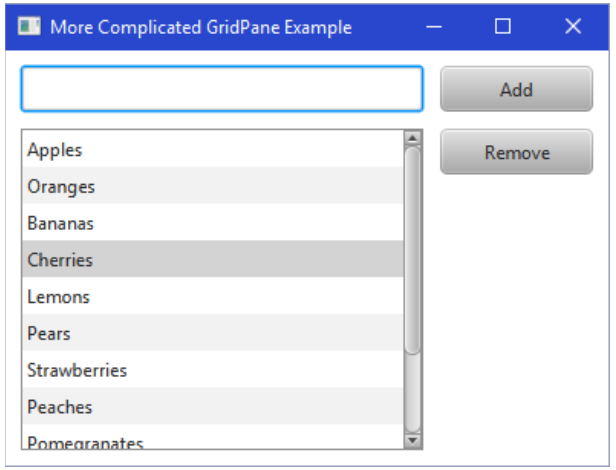
```
import javafx.application.Application; import javafx.collections.FXCollections;
import javafx.geometry.*; import javafx.scene.Scene;
import javafx.scene.control.*; import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class MoreComplicatedGridPaneExample extends Application {
      public void start(Stage primaryStage) {
            GridPane aPane = new GridPane();
            aPane.setPadding(new Insets(10, 10, 10, 10));
            TextField newItemField = new TextField();
            newItemField.setMinHeight(30); aPane.add(newItemField,0,0);
            aPane.setMargin(newItemField, new Insets(0, 0, 10, 0));
            Button addButton = new Button("Add"); aPane.add(addButton,1,0);
            addButton.setMinHeight(30); addButton.setMinWidth(100);
            aPane.setMargin(addButton, new Insets(0, 0, 10, 10));
            ListView<String> fruitList = new ListView<String>();
            String[] fruits = {"Apples", "Oranges", "Bananas", "Cherries",
                  "Lemons", "Pears", "Strawberries", "Peaches", "Pomegranates",
                  "Nectarines", "Apricots"};
```

Complete Code

```
        fruitList.setItems(FXCollections.observableArrayList(fruits));
        fruitList.setPrefWidth(Integer.MAX_VALUE);
        fruitList.setPrefHeight(Integer.MAX_VALUE);
        aPane.add(fruitList,0,1);
        Button removeButton = new Button("Remove");
        aPane.add(removeButton,1,1);
        removeButton.setMinHeight(30);
        removeButton.setMinWidth(100);
        aPane.setMargin(removeButton, new Insets(0, 0, 0, 10));
        aPane.setValignment(removeButton, VPos.TOP);
        primaryStage.setTitle("More Complicated GridPane Example");
        primaryStage.setScene(new Scene(aPane, 420,320));
        primaryStage.show();
    }
    public static void main(String[] args) {
            launch(args);
    }
}
```


More Com...
Add
Remove
Apples
Oranges
Bananas
Cherries
Lemons
Pears
Strawberries
Peaches
Pomegranates
Nectarines
Apricots


More Compli...
Add
Remove
Apples
Oranges
Bananas
Cherries
Lemons
Pears


More Complicated GridPane Example
Add
Remove
Apples
Oranges
Bananas
Cherries
Lemons
Pears
Strawberries
Peaches
Pomegranates