Application Program Development

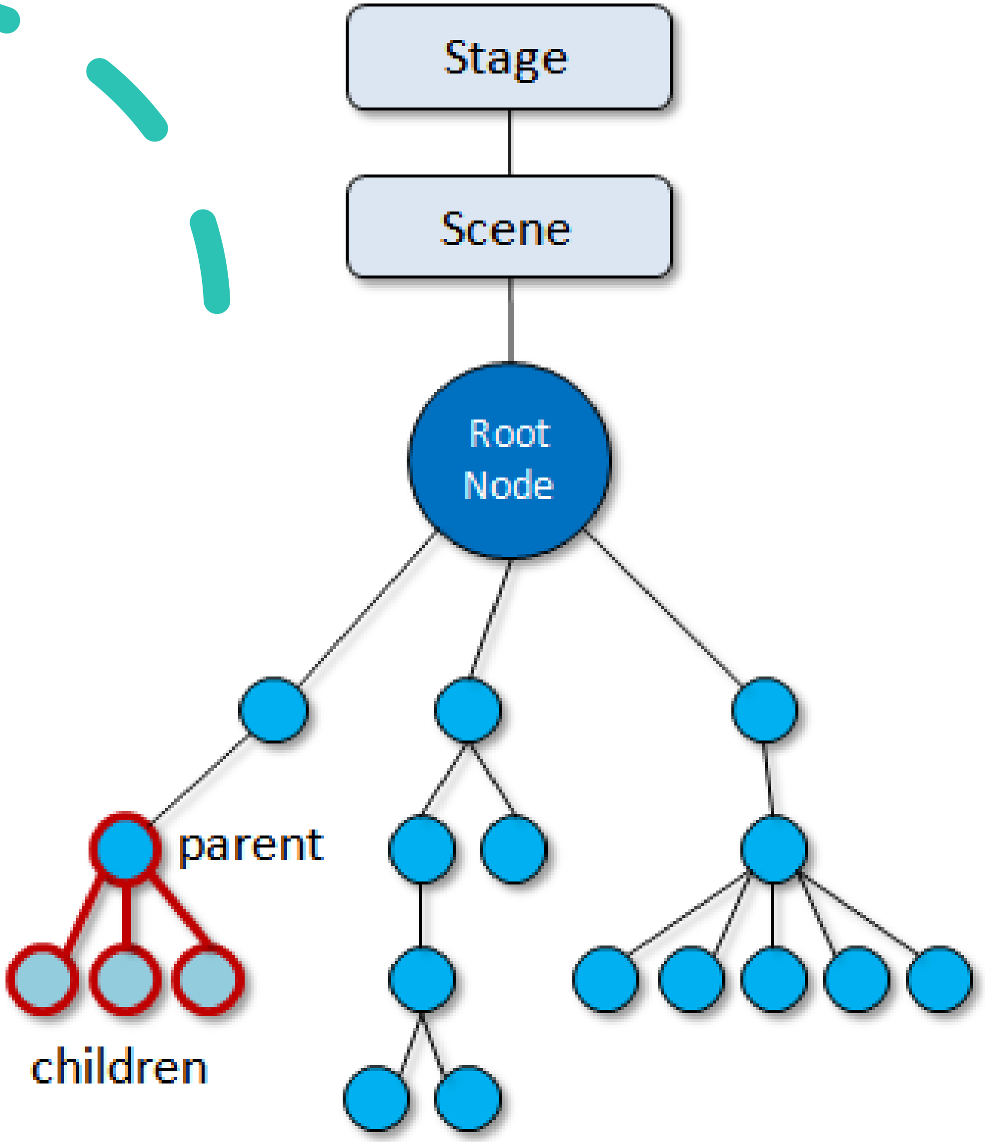
Segment : GUI with JavaFx

Outcomes

- Understanding Components in JavaFX.
- Understanding Containers in JavaFX.

Scene and Scene Graph

- The scene is arranged as a tree-like graph containing **Node** objects, which is called the ***scene graph***.
- The scene graph begins with a ***root node***.
- The nodes of the graph are basically the components of the window as well as the layout regions and the components within them.
- If node **c** is contained within another node **p**, then c is called a ***child*** of **p** and **p** is the ***parent*** of **c**.

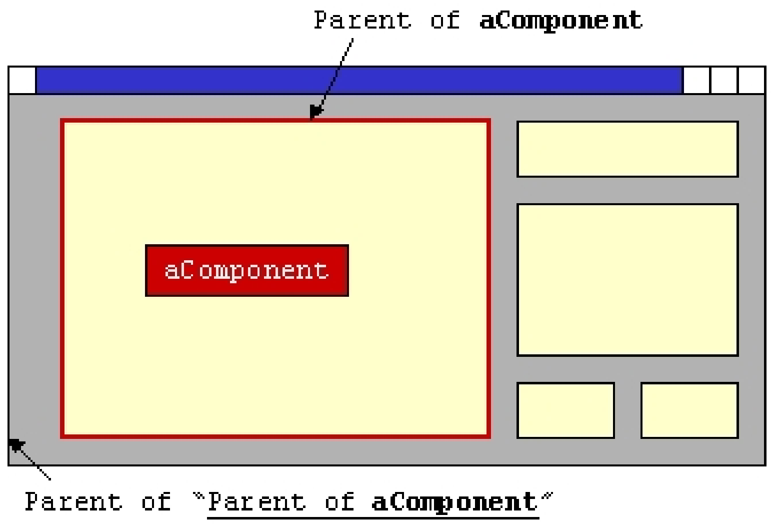


Window Component

- All window components actually keep pointers to their ***parent***. Parents of nested
- components are stored recursively:

```
Control aComponent;
Parent p;
Parent parentOfParent;
```

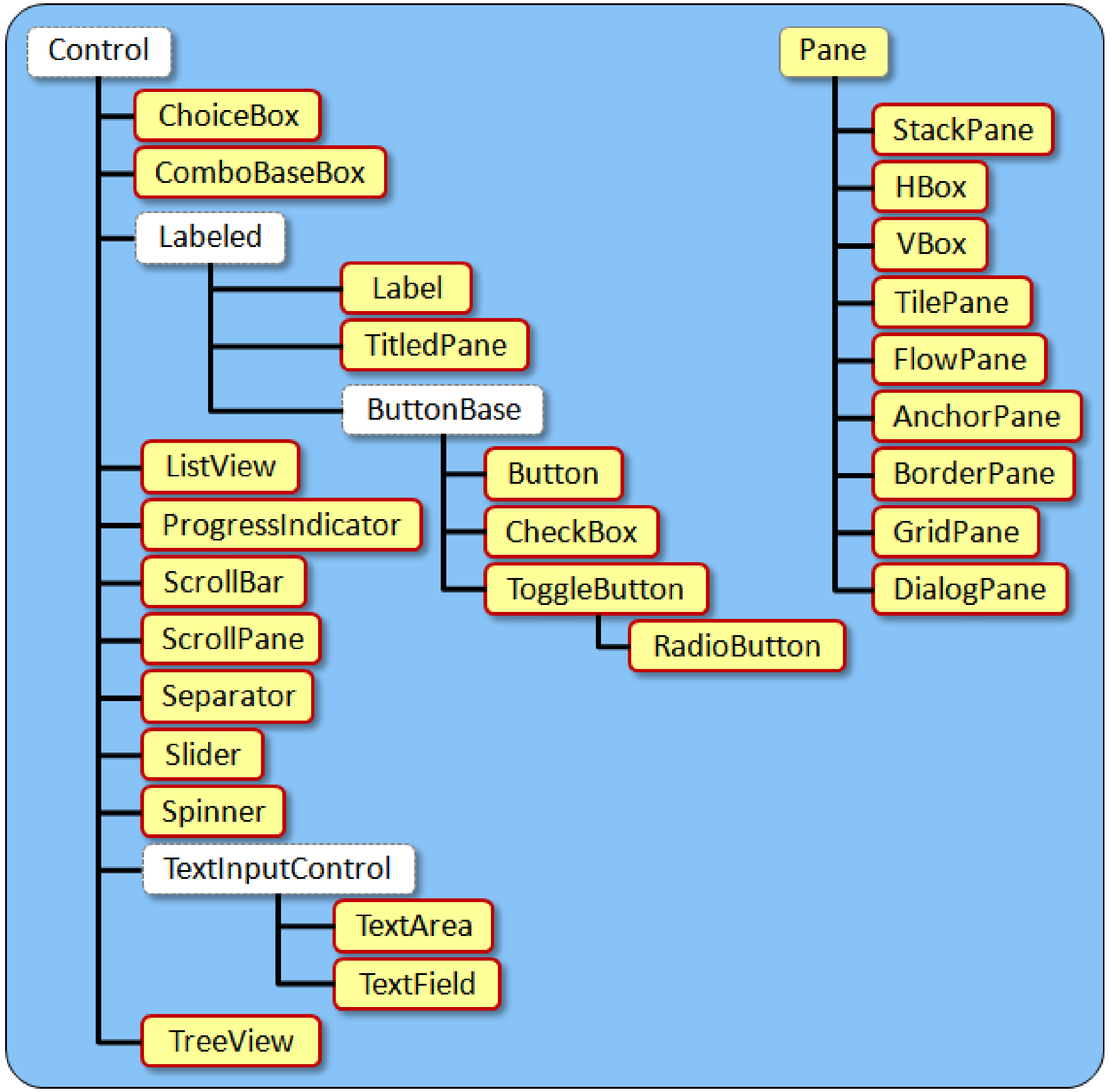
```
aComponent = ... ;
p = aComponent.getParent();
parentOfParent = p.getParent();
```



- Parents keep pointers to their children, and we can access these using the parent's **getChildren()** method:

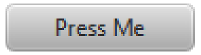
```
Parent aParent = ... ;
ObservableList<Node> children = aParent.getChildren();
Node c1 = children.get(0);
Node c2 = children.get(1);
Node c3 = children.get(2);
```

Different window components and Layouts



- Consider a simple button. We can create a button with some text on it as follows:

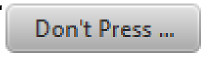
```
new Button("Press Me");
```



- We can store it in a variable as well.

```
Button b = new Button("Press Me");
```

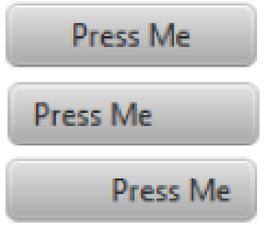
- You can change the text at any time by calling the **setText()** method, but depending on the size of the button, the text could be cut off



```
b.setText("Don't Press Me");
```

- You can even adjust alignment of the text by using the **setAlignment()** method

```
b.setAlignment(Pos.CENTER);

b.setAlignment(Pos.CENTER_LEFT);

b.setAlignment(Pos.CENTER_RIGHT);
```

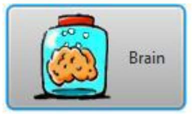


- This will require you to import the **Pos** class at the top of your code:

```
import javafx.geometry.Pos;
```

- You may even add an image to a component when created, as follows:

```
Image anImage = new Image(getClass().getResourceAsStream("brain.gif"));
Button b = new Button("Brain", new ImageView(anImage));
```

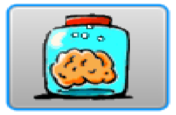


- This will require us to import these classes at the top of our code:

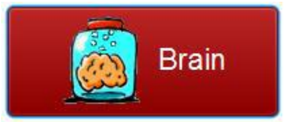
```
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
```

- So you will want to have the image file in the **src** folder that contains your code so that it can be found upon startup.
- If you don't want the text ... just the image, then you can use the **setGraphic()** method as follows:

```
Image anImage = new Image(getClass().getResourceAsStream("brain.gif"));
Button b = new Button();
b.setGraphic(new ImageView(anImage));
```



- Each component also has a variety of style settings that we can set.
- We will just look at three here:
  - text color
  - background color and
  - font.



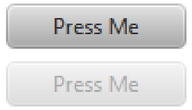
- We can use the setStyle() method to set any of these styles as follows:

```
b.setStyle("-fx-font: 22 arial; -fx-base: rgb(170,0,0); -fx-text-fill: rgb(255,255,255);");
```

- Notice that the method takes a string parameter. In that string we specify a sequence of ***style tags*** followed by their values. The **-fx-font:** tag allows us to specify a font size followed by a font type. In this case, we are selecting the **22pt Arial** font. The **-fx-base:** and **-fx-text-fill:** tags allow us to specify the background color and text color, respectively.

- It contains some style tag explanations, a list of predefined named colors and many other interesting details about styles. There are many interesting style settings.
- We can also ***disable*** components so that they cannot be controlled by the user:

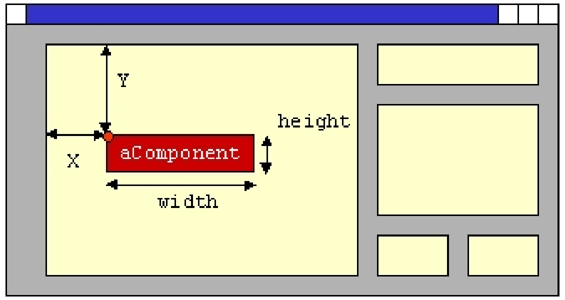
```
b.setDisable(false);

b.setDisable(true);
```



- By default, all components are enabled when created. At any time, we can also ***hide*** a component completely as follows:

```
b.setVisible(false);
```

- This will simply make the component invisible ... unable to be seen nor controlled by the user.
- All window components have an (x, y) location as well as width and height dimensions. The location is the (x,y) coordinate of the top/left of the component. It is a coordinate within the coordinate system defined by the top left corner of the parent node (similar to the position of a piece of paper on a bulletin board)

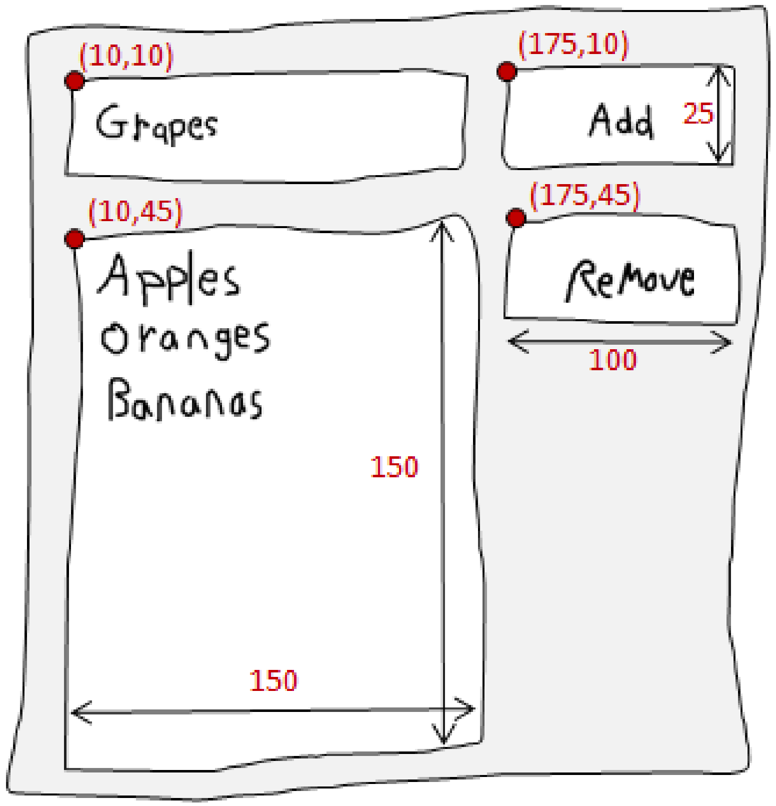


- Once we create the button, we can define how big we want it to be (i.e., it's preferred size) and where we want it to be located on the window by using these methods:

-
  ```
  b.relocate(40, 60); // x and y position with respect to its
  parent b.setPrefSize(100, 25); // width and height
  ```
- There are more attributes that we can set for components and we will investigate some more of them throughout the course.

Example

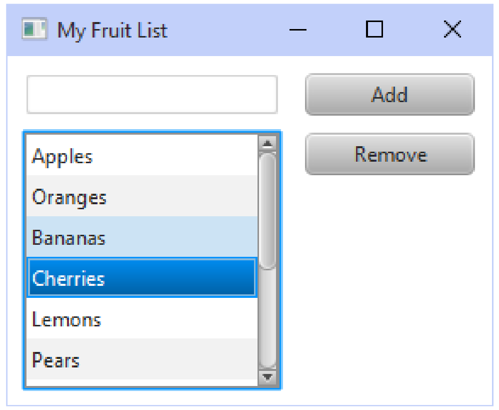
- Consider writing a program that creates the window shown in this rough diagram →
- The top/left component is a **TextField**. The bottom/left component is a **ListView** and the right two components are **Button** objects to add and remove items to the list.
- It is always a good idea to draw out a rough version of your user interface, identifying the top/left corner or all components as well as their dimensions and the width or all margins around the window border and between the components.
- To write the code, we need to create each of the window components. Here is the code to produce this window



```java
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ListView;
import javafx.scene.control.TextField;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;

public class FruitListApp extends Application {
 public void start(Stage primaryStage) {
  Pane aPane = new Pane(); // Create and position the "new item" TextField
  TextField newItemField = new TextField();
  newItemField.relocate(10, 10);
  newItemField.setPrefSize(150, 25); // Create and position the "Add" Button
  Button addButton = new Button("Add");
  addButton.relocate(175, 10);
  addButton.setPrefSize(100, 25); // Create and position the "Remove" Button
  Button removeButton = new Button("Remove");
  removeButton.relocate(175, 45);
  removeButton.setPrefSize(100, 25);
```

```
        // Create and position the "fruit" ListView with some fruits in it
        ListView<String> fruitList = new ListView<String>(); String[] fruits =
                            {"Apples", "Oranges", "Bananas", "Cherries",
                            "Lemons", "Pears", "Strawberries", "Peaches",
                            "Pomegranates", "Nectarines", "Apricots"};
        fruitList.setItems(FXCollections.observableArrayList(fruits));
        fruitList.relocate(10, 45);
        fruitList.setPrefSize(150, 150); // Add all the components to the window
        aPane.getChildren().addAll(newItemField, addButton, removeButton,
                                                                fruitList);
        primaryStage.setTitle("My Fruit List"); // Set title of window
        primaryStage.setScene(new Scene(aPane, 285,205)); // Set size of window
        primaryStage.show();
        }
        public static void main(String[] args) { launch(args); }
}
```

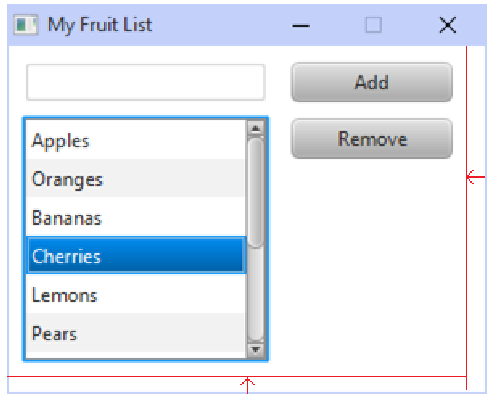
- Here is the window that is produced:



- We can also make the window non-resizable so that it stays at this size, since the components are not stretched anyway. To do this, we simply do this before we show the window:

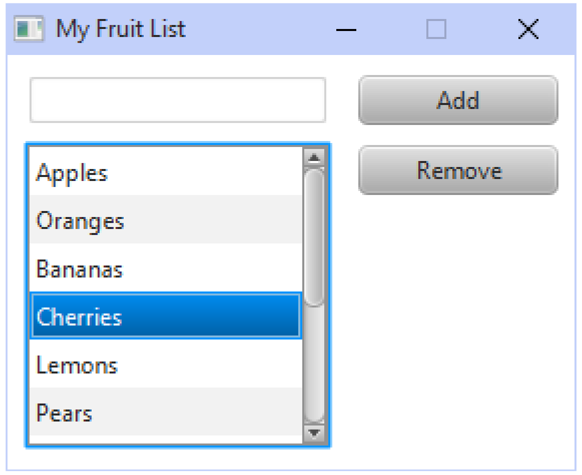
```
primaryStage.setResizable(false);
```

- You will notice, however, that the window is slightly bigger (i.e., 12 x 8 pixels bigger). This messes up the symmetrical look of our window, so we would need to adjust the size by decreasing the width by 12 and height by 8:



```
primaryStage.setScene(new Scene(aPane, 273,197));
```

- Below is the final result:


My Fruit List
Add
Remove
Apples
Oranges
Bananas
Cherries
Lemons
Pears