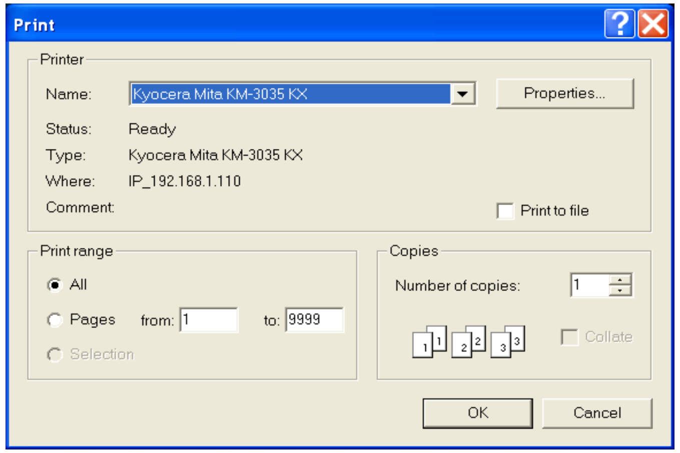
JavaFX

Topics

- GUI Programming with JavaFX
- Event driven programming

Graphical User Interface (GUI)

- Provides user-friendly human interaction
- History of GUI programming in Java
  - Abstract Window Toolkit (AWT)
  - Swings
  - JavaFX


Print

Printer

Name: Kyocera Mita KM-3035 KX | Properties...

Status: Ready
Type: Kyocera Mita KM-3035 KX
Where: IP_192.168.1.110
Comment:

☐ Print to file

Print range

[✓ All]
Pages from: 1 to: 9999
Selection

Copies

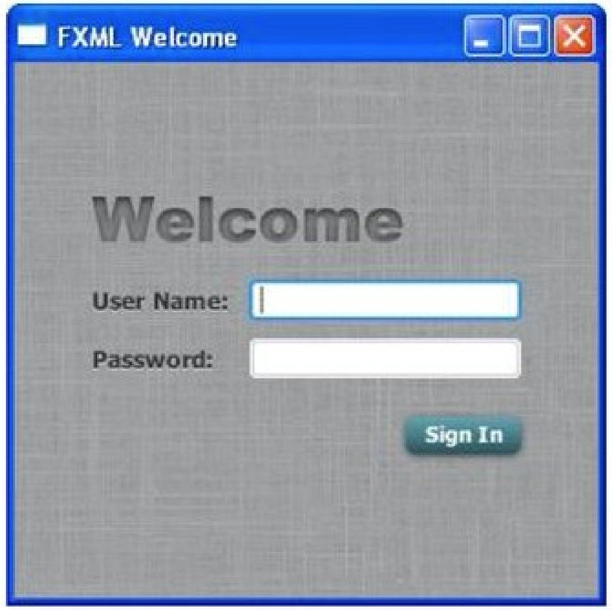
Number of copies: 1

☐ Collate

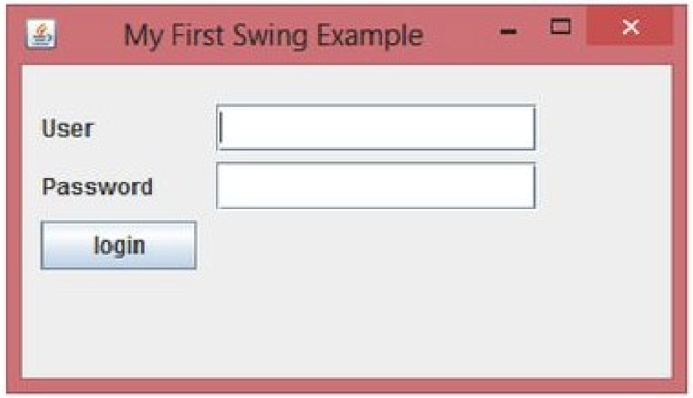
OK | Cancel

JavaFX vs Swings vs AWT

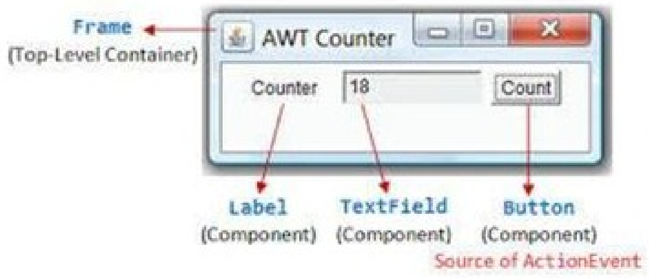
JavaFX example



Swing example



AWT example

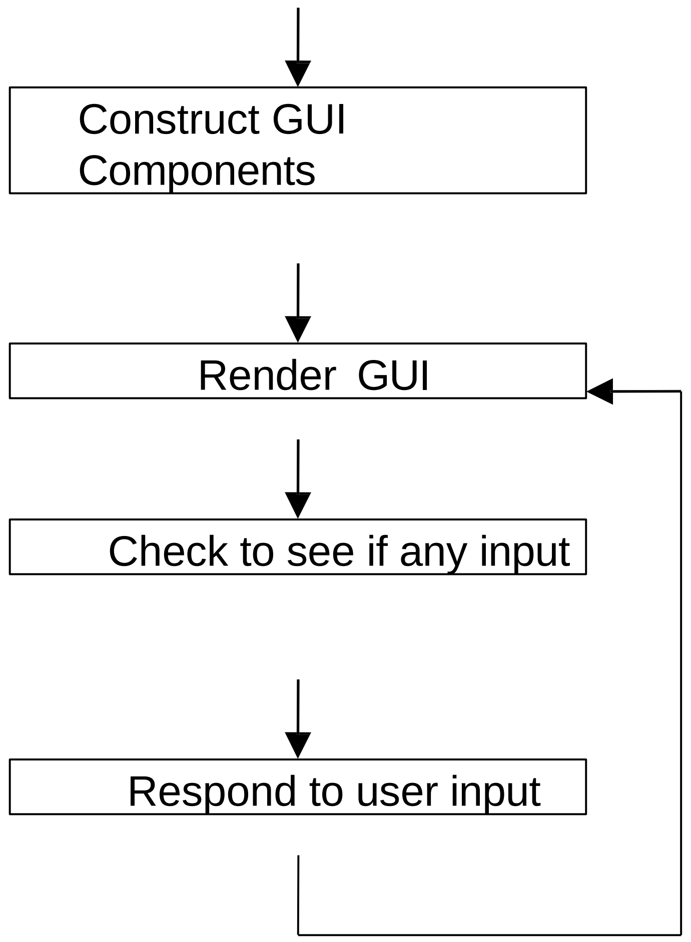


How do GUIs work?

- GUIs loop and respond to events:
  - Example: a mouse click on a button

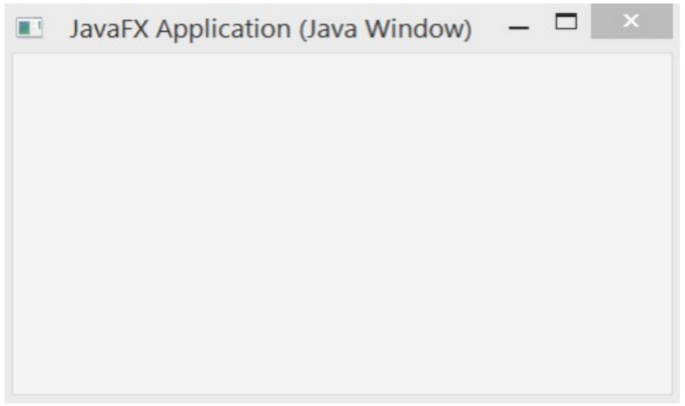
  - The Operating System recognizes mouse click, determines which window it was inside and notifies that program by putting the event on that program's input buffer/queue

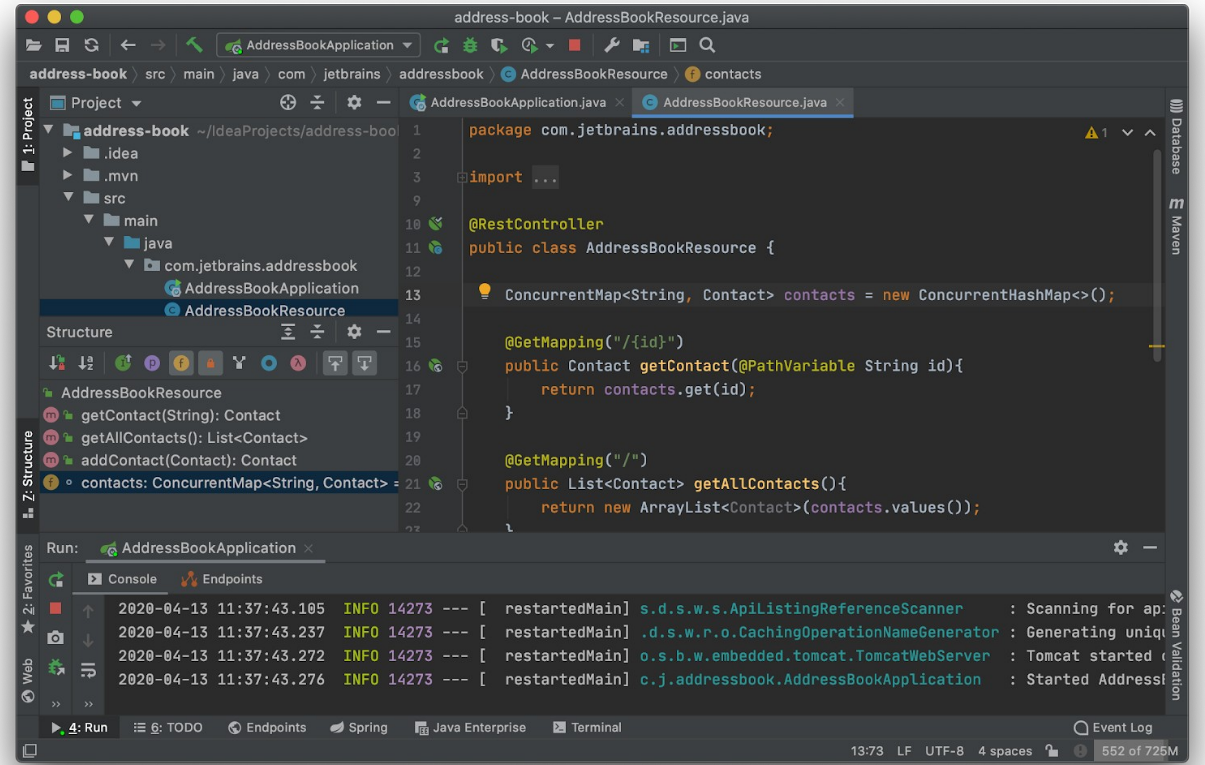
  - The program runs in loop:
    - renders the GUI
    - checks input buffer filled by OS
    - if it finds a mouse click, determines which component in the program, respond appropriately according to handler



How does GUI Framework Help?

- **Provides ready made visible, interactive, customizable components**
  - you wouldn’t want to have to code your own window
  - Imagine creating IntelliJ Idea GUI from scratch





```
package com.jetbrains.addressbook;

import ...

@RestController
public class AddressBookResource {

    ConcurrentMap<String, Contact> contacts = new ConcurrentHashMap<>();

    @GetMapping("/{id}")
    public Contact getContact(@PathVariable String id){
        return contacts.get(id);
    }

    @GetMapping("/")
    public List<Contact> getAllContacts(){
        return new ArrayList<Contact>(contacts.values());
```

```
2020-04-13 11:37:43.105  INFO 14273 --- [  restartedMain] s.d.s.w.s.ApiListingReferenceScanner     : Scanning for ap:
2020-04-13 11:37:43.237  INFO 14273 --- [  restartedMain] .d.s.w.r.o.CachingOperationNameGenerator : Generating uniqu
2020-04-13 11:37:43.272  INFO 14273 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started o
2020-04-13 11:37:43.276  INFO 14273 --- [  restartedMain] c.j.addressbook.AddressBookApplication   : Started AddressE
```

- JavaFX library simplifies the building of complex graphically rich client applications
- It provides simple APIs to add graphics, media, web content, UI controls etc., in the applications

How to install JavaFX

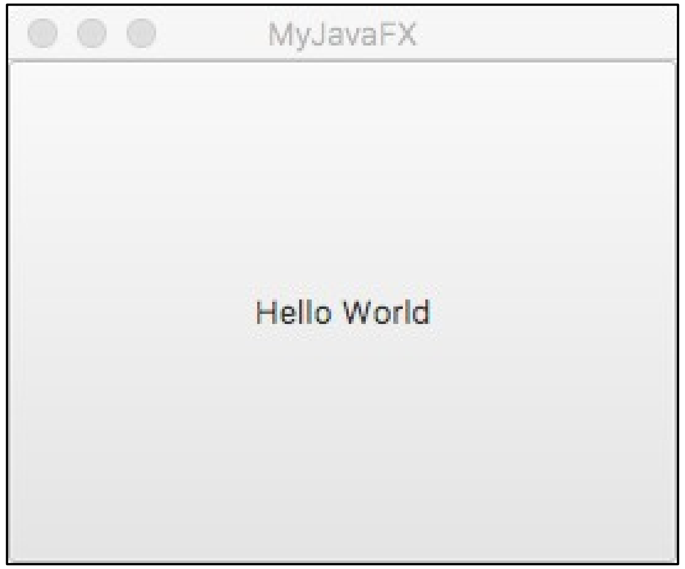
- Download the JavaFX modules: https://gluonhq.com/products/javafx/ and set up/install the modules in your IDE (i.e. IntelliJ Idea)
- Or, install a full jdk from bellsoft (Liberica JDK): https://bell-sw.com/pages/downloads/#jdk-17-lts
- You can use JavaFX by importing the modules:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
```

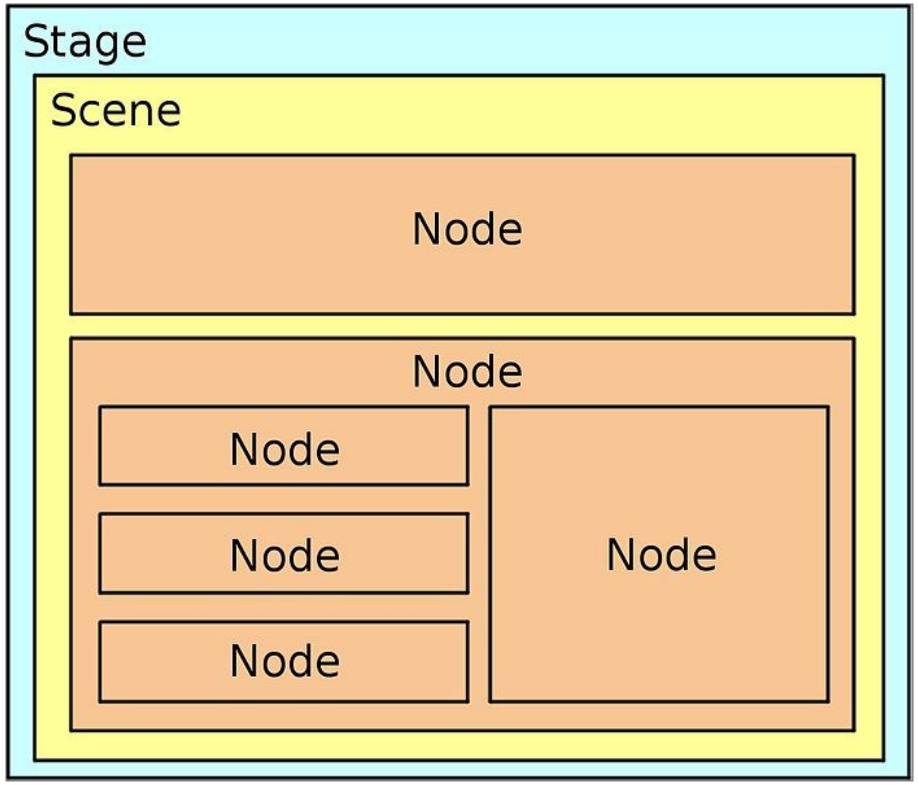
JavaFX: Hello World

```
public class HelloWorld extends Application { public
    static void main(String[] args) {
        launch(args);
    }

    //Override the start method in the Application class
    @Override
    public void start(Stage primaryStage) {
        // Set the stage title
        primaryStage.setTitle("MyJavaFX");
        // Create a button and place it in the scene
        Button btn = new Button(”Hello World");
        Scene scene = new Scene(btn, 200, 250);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage
        primaryStage.show();
    }

}
```

- The main class for a JavaFX application extends the `javafx.application.Application` abstract class
  - o The start() method is the main entry point for all JavaFX applications



Basic Structure of JavaFX

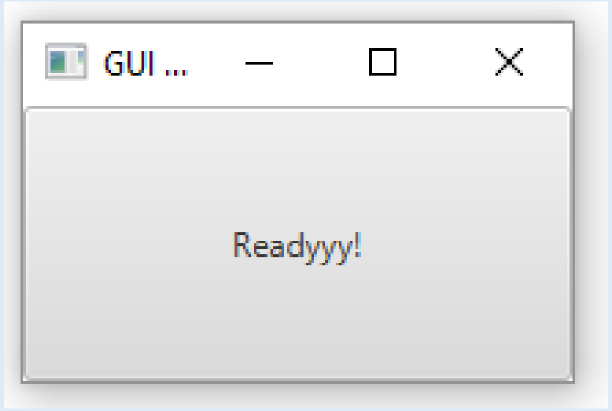


- Class `javafx.stage.Stage` is the top level JavaFX container
- Class `javafx.scene.Scene` class is the container for all content in a scene graph
- Abstract class `javafx.application.Application` is the entry point for JavaFX applications
  - Executes the user application and processes input events
  - User just need to Override the `start` method!
- Components can be created/added programmatically

```
Parent p; Node n;

p.getChildren().add(n)
```

Multiple stages

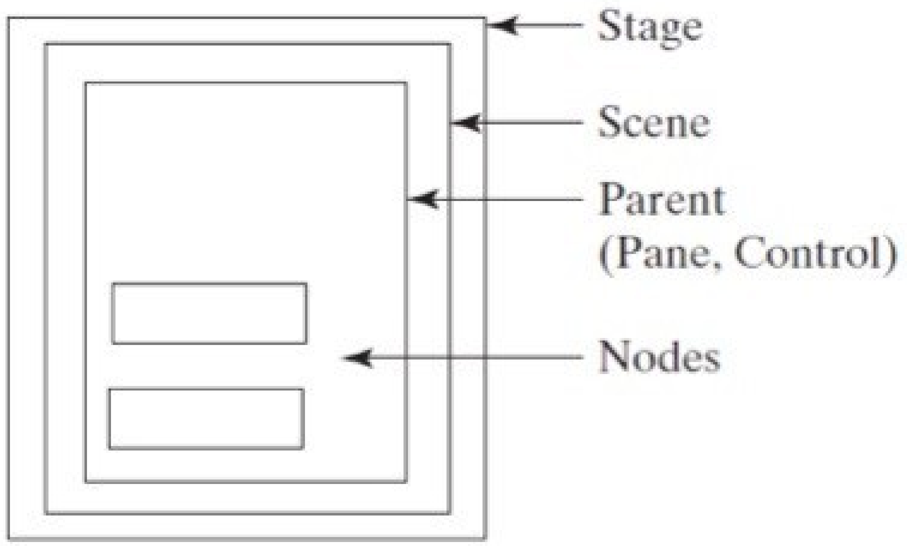
```
// ... other code parts follow previous code
public void start(Stage stg1) throws Exception {
    stg1.setTitle("GUI 002");
    stg1.setScene(new Scene(new Button("Gooo!"), 200,100));
    stg1.show();

    Stage stg2 = new Stage();
    stg2.setTitle("GUI 002");
    stg2.setScene(new Scene(new Button("Steadyyy!"), 200,100));
    stg2.show();

    Stage stg3 = new Stage();
    stg3.setTitle("GUI 002");
    stg3.setScene(new Scene(new Button("Readyyy!"), 200,100));
    stg3.show();
}
```

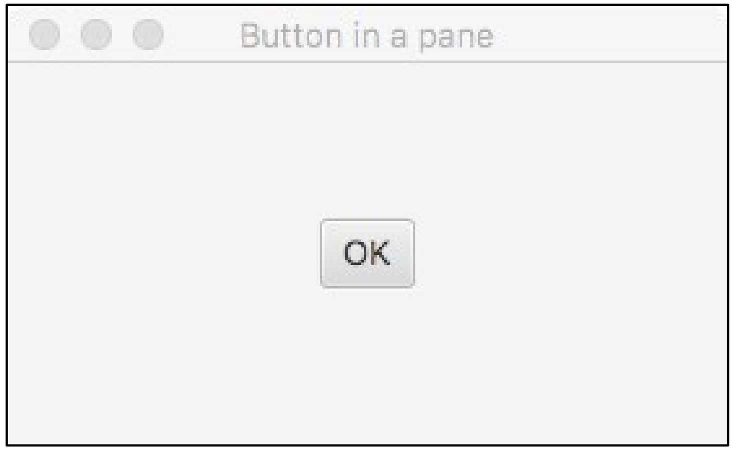


Close the window, you'll see the Steadyyy! stage,
close the window again, you'll see the Gooo! stage.

JavaFX: Button in a Pane

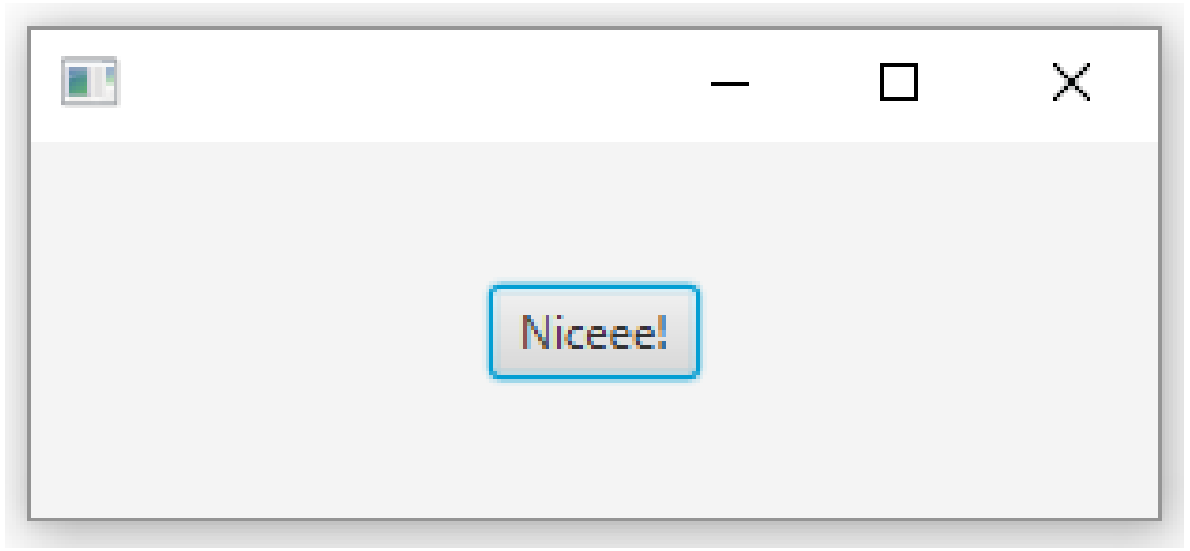
```
public class ButtonInPane extends Application { public
    static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        // Set the stage title primaryStage.setTitle(”Button
        in a Pane");
        // Create a button and place it in the scene
        Button btn = new Button(”OK");
        // Create a pane and place a button in the pane
        StackPane pane = new StackPane();
        pane.getChildren().add(btn);
        // Create scene with a pane inside it Scene scene =
        new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage primaryStage.show();
    }
}
```





A pane is like a container

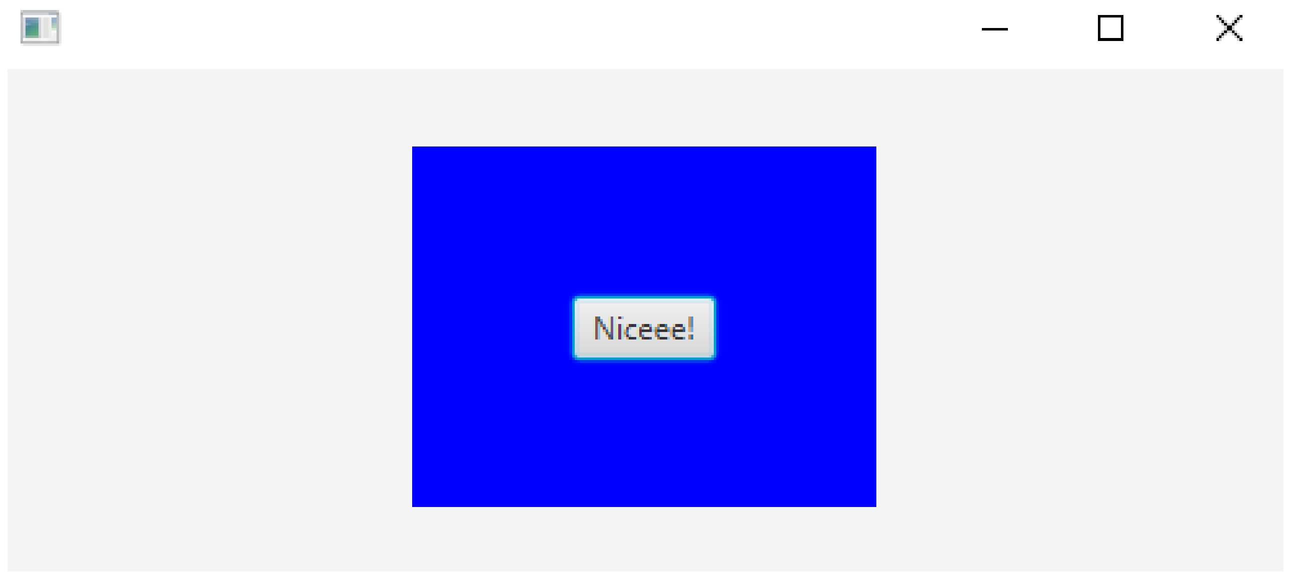
```
public void start(Stage stg1) throws Exception {
    StackPane pn = new StackPane( );
    pn.getChildren( ).add(new Button("Niceee!")); // because Nice! would sound rude
    Scene scn = new Scene(pn, 300, 100);
    stg1.setScene(scn);
    stg1.show( );
}
```



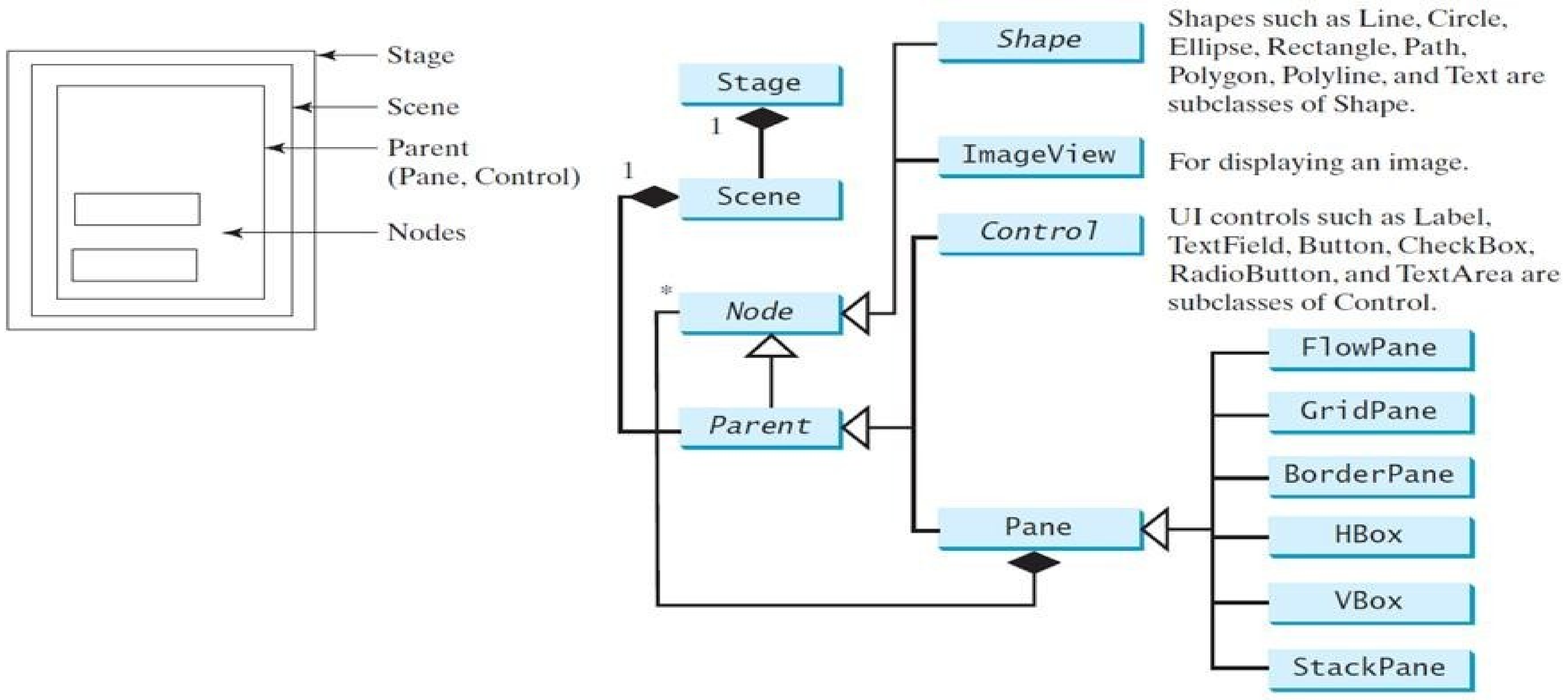
Note that the button now doesn't occupy the whole scene, thanks to Pane!

Stacked square and button

```
public void start(Stage stg1) throws Exception {
   Rectangle rect = new Rectangle(100, 100, 180, 140);
   rect.setFill(Color.BLUE);

   StackPane pn = new StackPane(rect, new Button("Niceee!"));
   Scene scn = new Scene(pn, 500, 200);
   stg1.setScene(scn);
   stg1.show( );
}
```



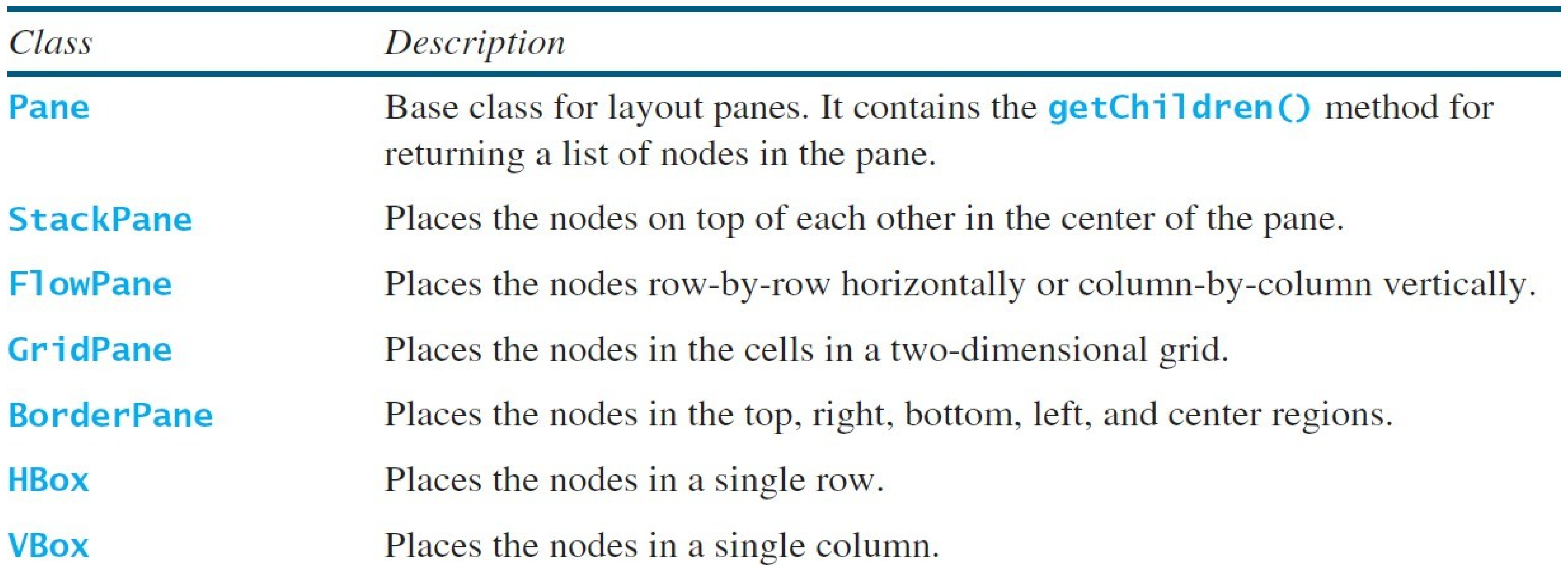
Panes, UI Controls, and Shapes



Stage
Scene
Parent
(Pane, Control)
Nodes
Stage
1
1
Scene
*
Node
Parent
Shape
ImageView
Control
Pane
FlowPane
GridPane
BorderPane
HBox
VBox
StackPane
Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.
For displaying an image.
UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.

Layout Panes

- JavaFX provides many **types of** **panes for organizing nodes** **in a container.**

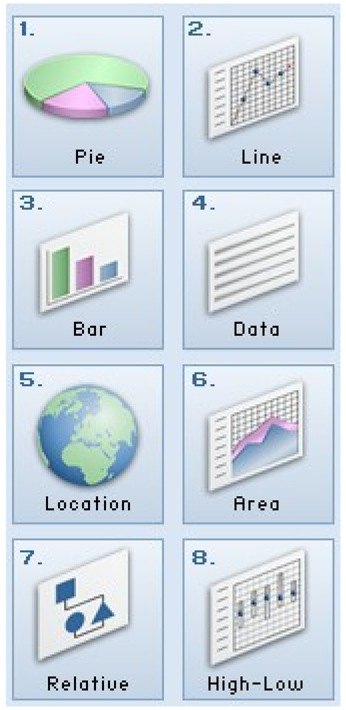


Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

FlowPane

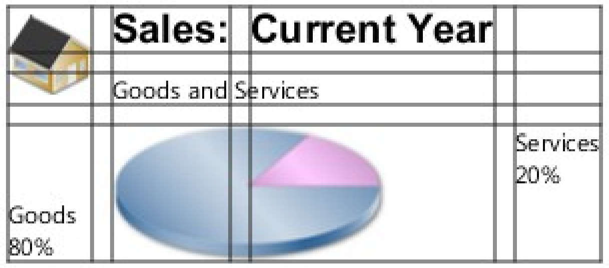
- The nodes within a FlowPane layout pane are laid out consecutively and wrap at the boundary set for the pane.
- Nodes can flow vertically (in columns) or horizontally (in rows).

```
public FlowPane addFlowPane() {
    FlowPane flow = new FlowPane();
    flow.setPadding(new Insets(5, 0, 5, 0));
    flow.setVgap(4);
    flow.setHgap(4);
    flow.setPrefWrapLength(170); // preferred width allows for two columns
    flow.setStyle("-fx-background-color: DAE6F3;");
    ImageView pages[] = new ImageView[8];
    for (int i=0; i<8; i++) {
        pages[i] = new ImageView(
            new Image(LayoutSample.class.getResourceAsStream(
            "graphics/chart_"+(i+1)+".png")));
        flow.getChildren().add(pages[i]);
    }
    return flow;
}
```



GridPane

- The GridPane layout pane enables you to create a flexible grid of rows and columns in which to lay out nodes.
- Nodes can be placed in any cell in the grid and can span cells as needed.
- A grid pane is useful for creating forms or any layout that is organized in rows and columns

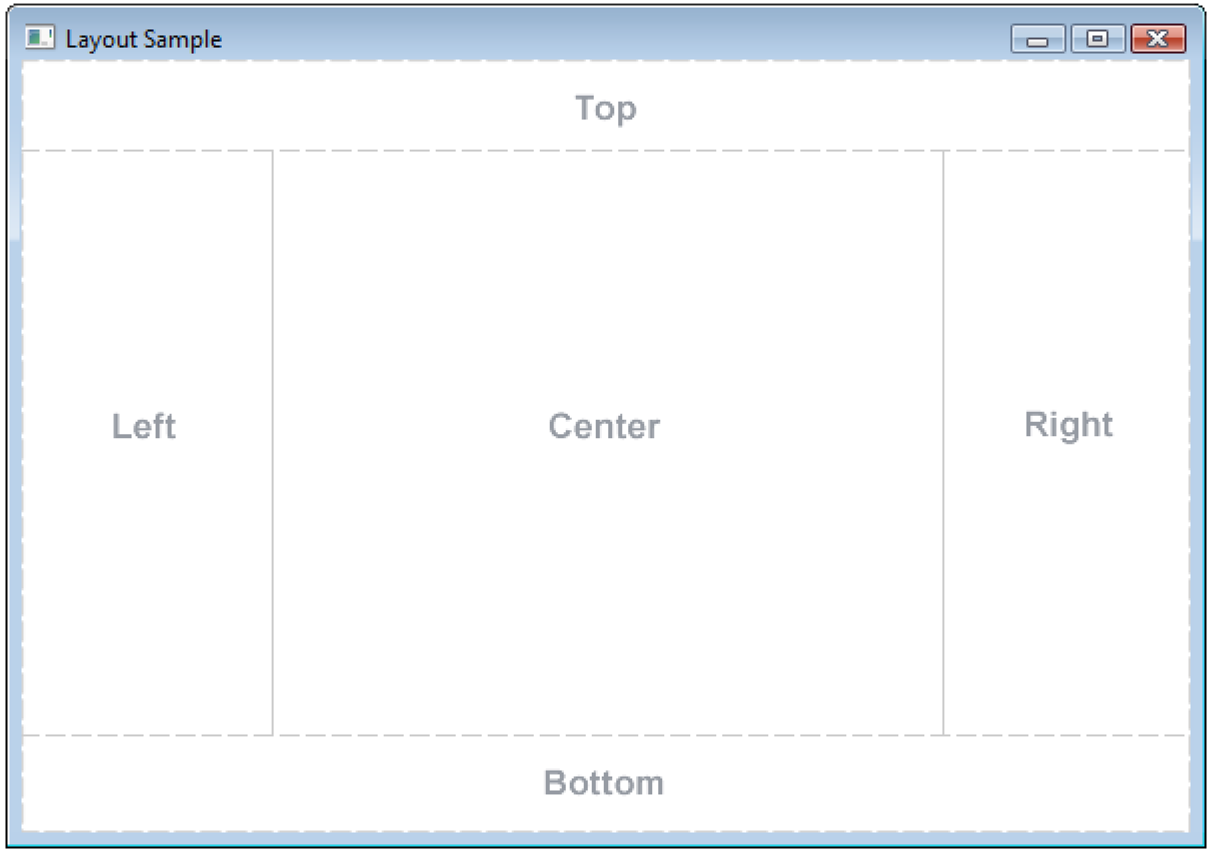


```
public GridPane addGridPane() {
    GridPane grid = new GridPane();
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(0, 10, 0, 10));
    // Category in column 2, row 1
    Text category = new Text("Sales:");
    category.setFont(Font.font("Arial", FontWeight.BOLD, 20));
    grid.add(category, 1, 0);
    // Title in column 3, row 1
    Text chartTitle = new Text("Current Year");
    chartTitle.setFont(Font.font("Arial", FontWeight.BOLD, 20));
    grid.add(chartTitle, 2, 0);
    // Subtitle in columns 2-3, row 2
    Text chartSubtitle = new Text("Goods and Services");
    grid.add(chartSubtitle, 1, 1, 2, 1);
    // House icon in column 1, rows 1-2
    ImageView imageHouse = new ImageView(
      new Image(LayoutSample.class.getResourceAsStream("graphics/house.png")));
    grid.add(imageHouse, 0, 0, 1, 2);
    // Left label in column 1 (bottom), row 3
    Text goodsPercent = new Text("Goods\n80%");
    GridPane.setValignment(goodsPercent, VPos.BOTTOM);
    grid.add(goodsPercent, 0, 2);
    // Chart in columns 2-3, row 3
    ImageView imageChart = new ImageView(
     new Image(LayoutSample.class.getResourceAsStream("graphics/piechart.png")));
    grid.add(imageChart, 1, 2, 2, 1);
    // Right label in column 4 (top), row 3
    Text servicesPercent = new Text("Services\n20%");
    GridPane.setValignment(servicesPercent, VPos.TOP);
    grid.add(servicesPercent, 3, 2);
    return grid;
}
```

BorderPane

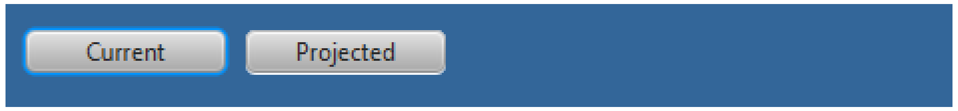
- The BorderPane layout pane provides five regions in which to place nodes: top, bottom, left, right, and center.

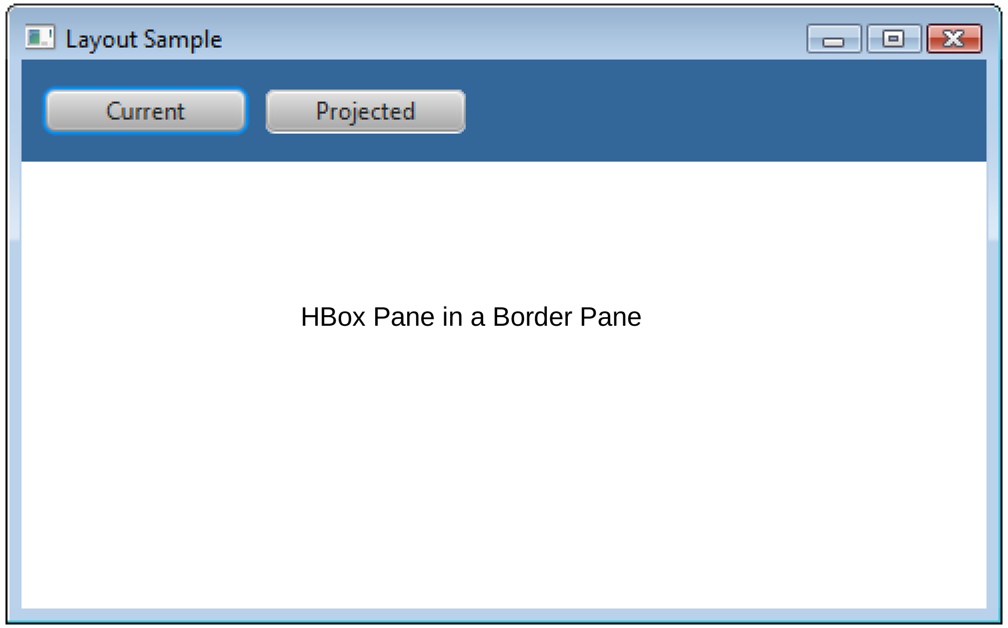
```
BorderPane border = new BorderPane();
HBox hbox = addHBox()
border.setTop(hbox);
border.setLeft(addVBox());
addStackPane(hbox);         // Add stack to HBox in top region
border.setCenter(addGridPane());
border.setRight(addFlowPane());
```



HBox

- The HBox layout pane provides an easy way for arranging a series of nodes in a single row.





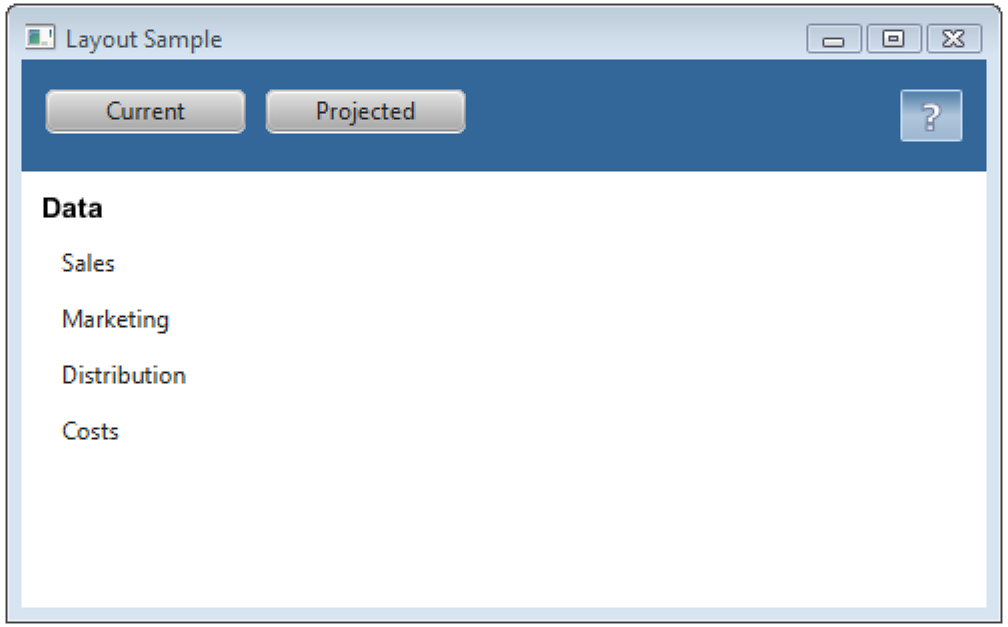
HBox Pane in a Border Pane

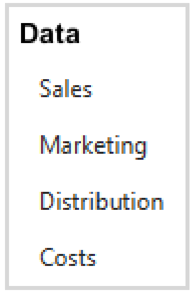
```
public HBox addHBox() {
    HBox hbox = new HBox();
    hbox.setPadding(new Insets(15, 12, 15, 12));
    hbox.setSpacing(10);
    hbox.setStyle("-fx-background-color: #336699;");
    Button buttonCurrent = new Button("Current");
    buttonCurrent.setPrefSize(100, 20);
    Button buttonProjected = new Button("Projected");
    buttonProjected.setPrefSize(100, 20);
    hbox.getChildren().addAll(buttonCurrent,
buttonProjected);
    return hbox;
}
```

VBox

- The VBox layout pane is similar to the HBox layout pane, except that the nodes are arranged in a single column.

```
public VBox addVBox(); {
    VBox vbox = new VBox();
    vbox.setPadding(new Insets(10));
    vbox.setSpacing(8);
    Text title = new Text("Data");
    title.setFont(Font.font("Arial", FontWeight.BOLD, 14));
    vbox.getChildren().add(title);
    Hyperlink options[] = new Hyperlink[] {
        new Hyperlink("Sales"),
        new Hyperlink("Marketing"),
        new Hyperlink("Distribution"),
        new Hyperlink("Costs")};
    for (int i=0; i<4; i++) {
        VBox.setMargin(options[i], new Insets(0, 0, 0, 8));
        vbox.getChildren().add(options[i]);
    }
    return vbox;
}
```

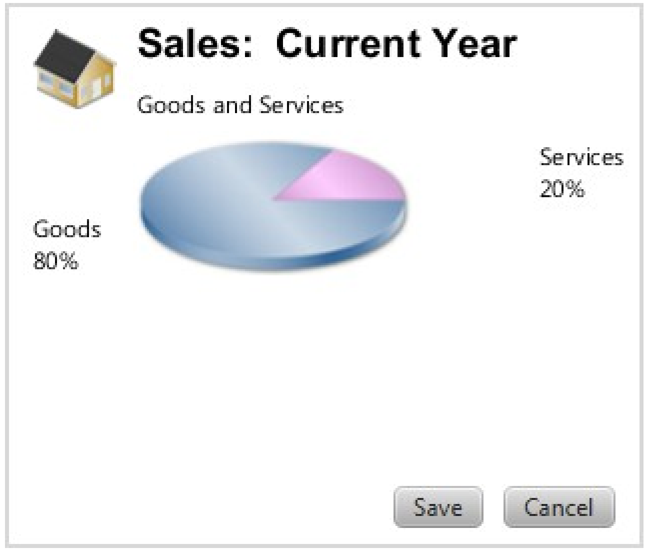




Hbox and VBox Pane in a Border Pane

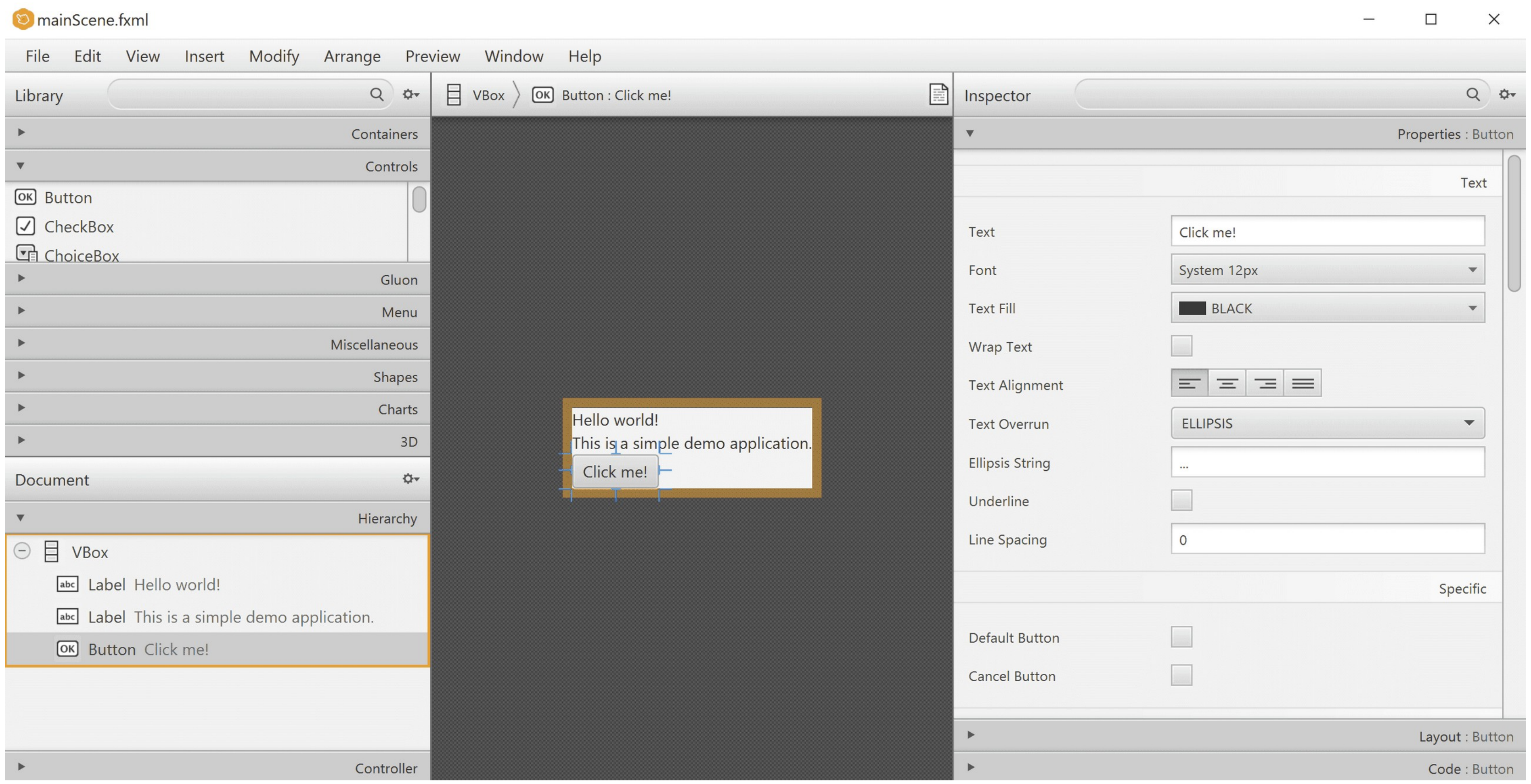
AnchorPane

- The AnchorPane layout pane enables you to anchor nodes to the top, bottom, left side, right side, or center of the pane.
- As the window is resized, the nodes maintain their position relative to their anchor point.
- Nodes can be anchored to more than one position and more than one node can be anchored to the same position.



```
public AnchorPane addAnchorPane(GridPane grid) {
    AnchorPane anchorpane = new AnchorPane();
    Button buttonSave = new Button("Save");
    Button buttonCancel = new Button("Cancel");
    HBox hb = new HBox();
    hb.setPadding(new Insets(0, 10, 10, 10));
    hb.setSpacing(10);
    hb.getChildren().addAll(buttonSave, buttonCancel);
    anchorpane.getChildren().addAll(grid,hb);   // Add grid from GridPane example
    AnchorPane.setBottomAnchor(hb, 8.0);
    AnchorPane.setRightAnchor(hb, 5.0);
    AnchorPane.setTopAnchor(grid, 10.0);
    return anchorpane;
}
```

Alternatively: Scene Builder + FXML



VBox
- Label Hello world!
- Label This is a simple demo application.
- Button Click me!

Text

Text	Click me!
Font	System 12px
Text Fill	BLACK
Wrap Text	☐
Text Alignment	
Text Overrun	ELLIPSIS
Ellipsis String	...
Underline	☐
Line Spacing	0

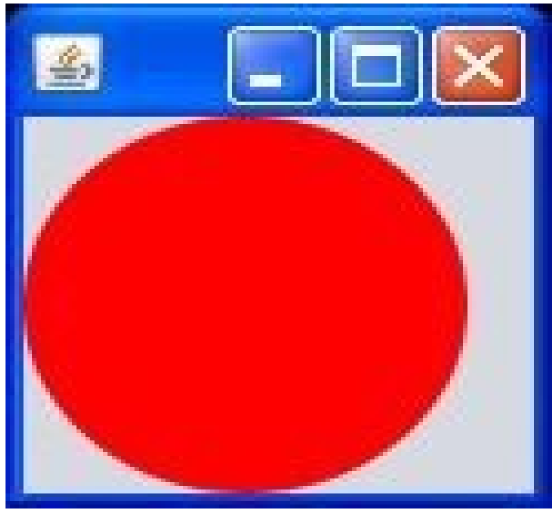
Specific

Default Button	☐
Cancel Button	☐

- Scene Builder provides a graphical interface for designing and constructing user interfaces
- Scene Builder allows for components to be created, placed, and for many of their properties to be modified
- Saves your layout in an FXML file, which could be read in the Java file to create the GUI

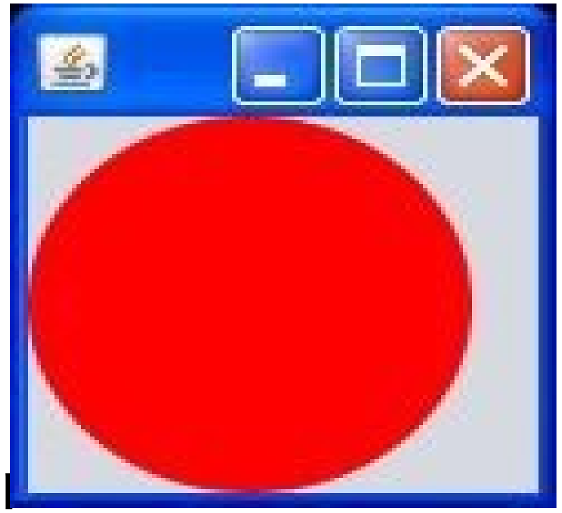
Let’s Compare: JavaFX 2.0

```
public class JavaFXTest extends Application
  { @Override public void start(Stage stage) {
    stage.setTitle(“FXML Example”);
    Group root = new Group();
    Scene scene = new Scene(root,100,100);
    stage.setScene(scene);

    Circle c1 =
      new Circle(50.0f, 50.0f, 50.0f,
      Color.RED);

    root.getChildren().add(c1);
    stage.setVisible(true); stage.show();
  }

  public static void main(String[] args)
     { launch(args);
  }
}
```



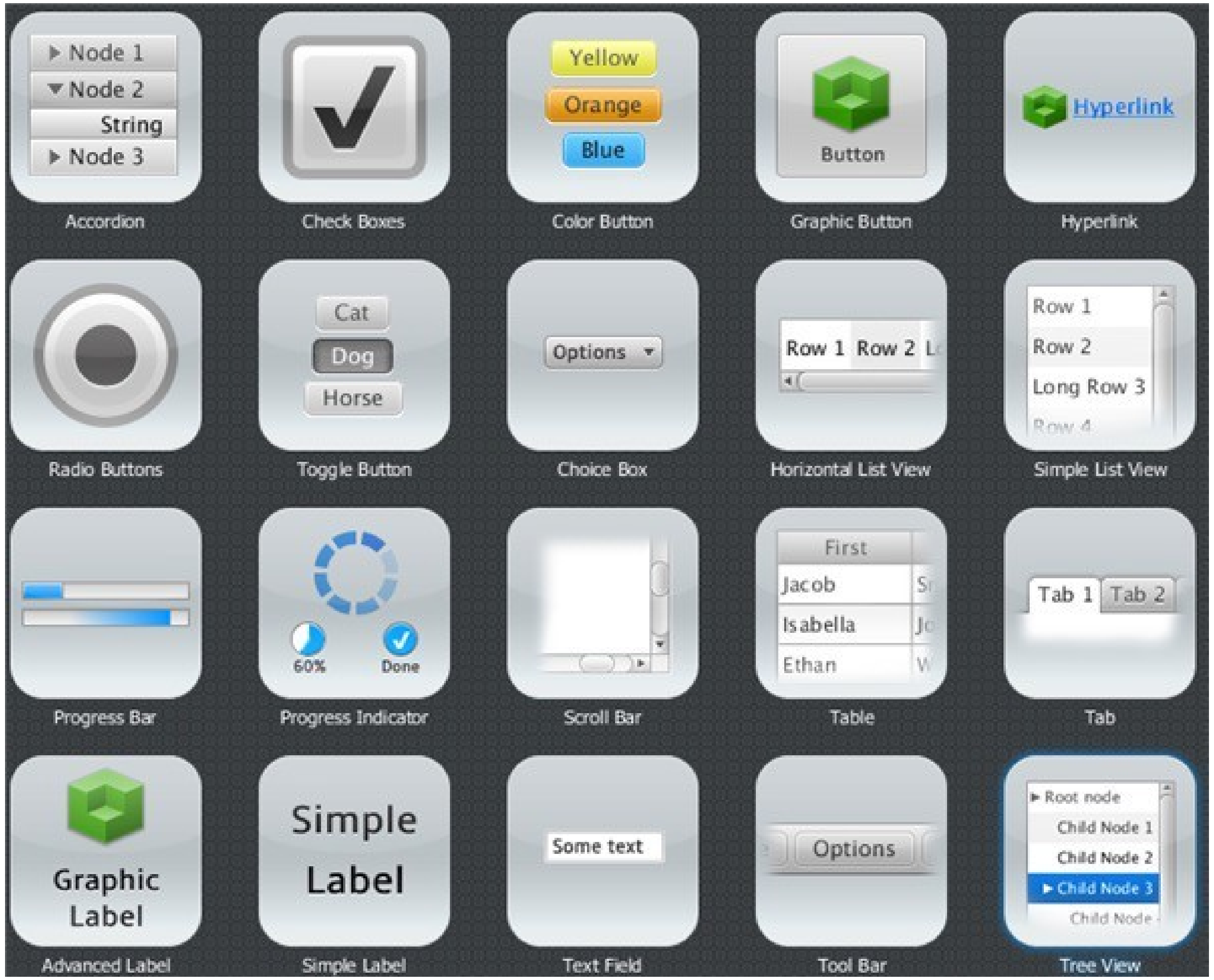
Let’s Compare: FXML

```
<BorderPane>
<center>
<Circle radius=“50” centerX=“50” centerY=“50”/>
</center>
</BorderPane>
```

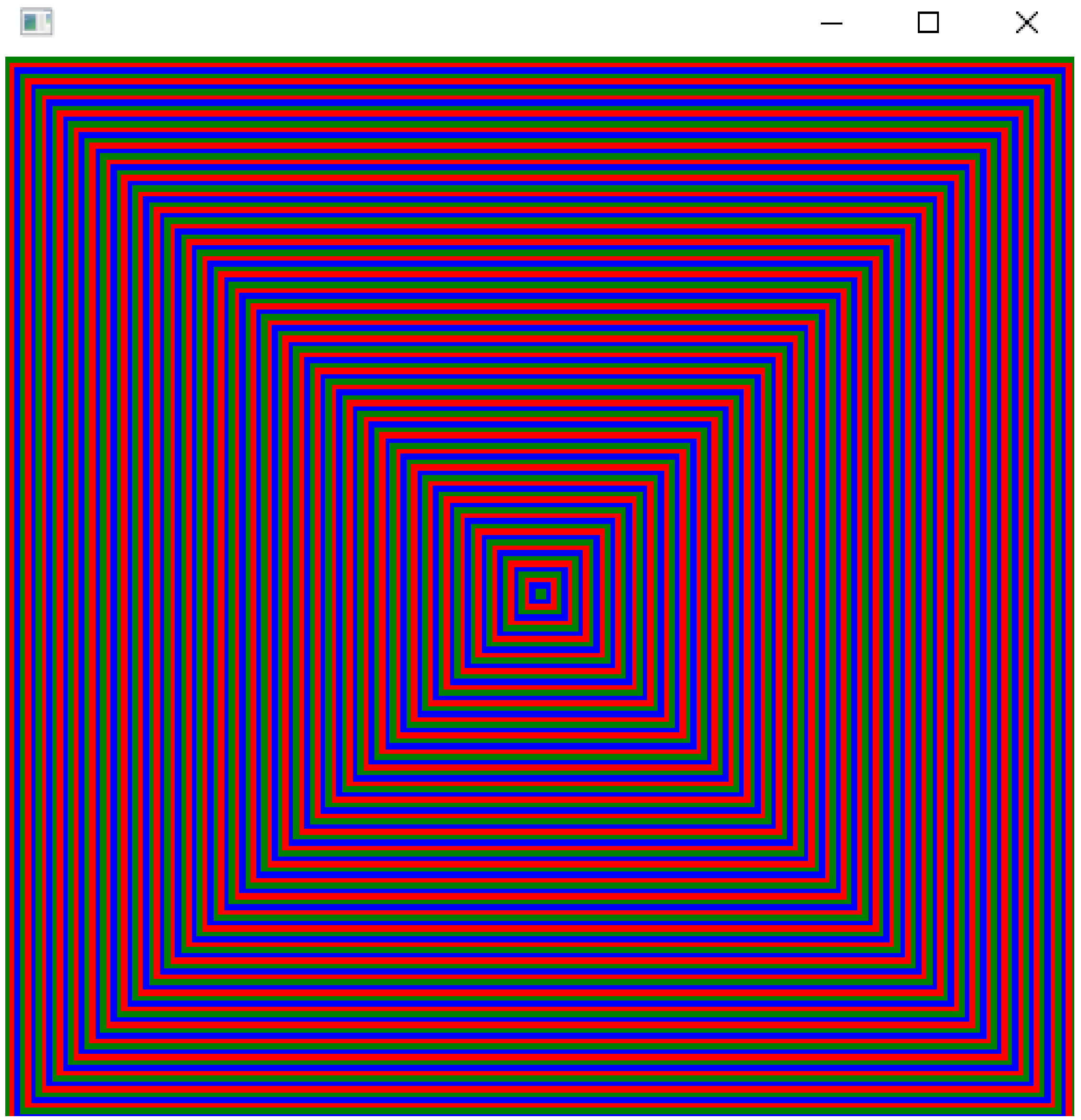


```
public class JavaFXTest extends Application { @Override
  start(Stage stage) {
    stage.setTitle(“FXML Example”);
    Parent root = FXMLLoader.load(getClass().getResource(“example.fxml");
    stage.setScene(new Scene(root));
    stage.show();

  }
  public static void main(String[] args) { launch(args);
  }
}
```

JavaFX UI Controls



Example: Java art



Java art

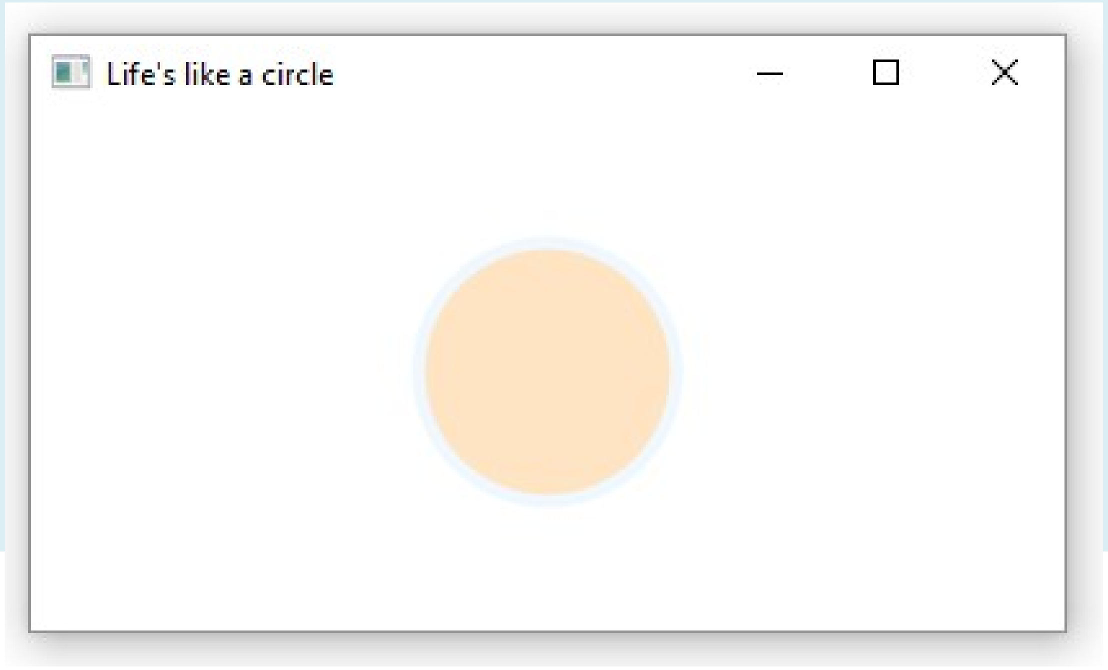
```
public void start(Stage stg1) throws Exception {
        StackPane pn = new StackPane();
        for(int i = 100; i > 0; i--) {
                Rectangle rect = new Rectangle(0,0,5*i,5*i);
                switch(i % 3) {
                        case 0:
                                rect.setFill(Color.RED); break;
                        case 1:
                                rect.setFill(Color.GREEN); break;
                        case 2:
                                rect.setFill(Color.BLUE); break;
                }
                pn.getChildren().add(rect);
        }

        Scene scn = new Scene(pn, 500, 500);
        stg1.setScene(scn);
        stg1.show();
}
```

Circle

```
public void start(Stage stg1) throws Exception {
      Circle c = new Circle();
      c.setCenterX(200);
      c.setCenterY(100);
      c.setRadius(50);
      c.setStroke(Color.ALICEBLUE);
      c.setStrokeWidth(5);
      c.setFill(Color.BISQUE);

      Pane pn = new Pane();
      pn.getChildren().add(c);
      Scene scn = new Scene(pn, 400, 200);
      stg1.setTitle("Life's like a circle");
      stg1.setScene(scn);
      stg1.show();
}
```

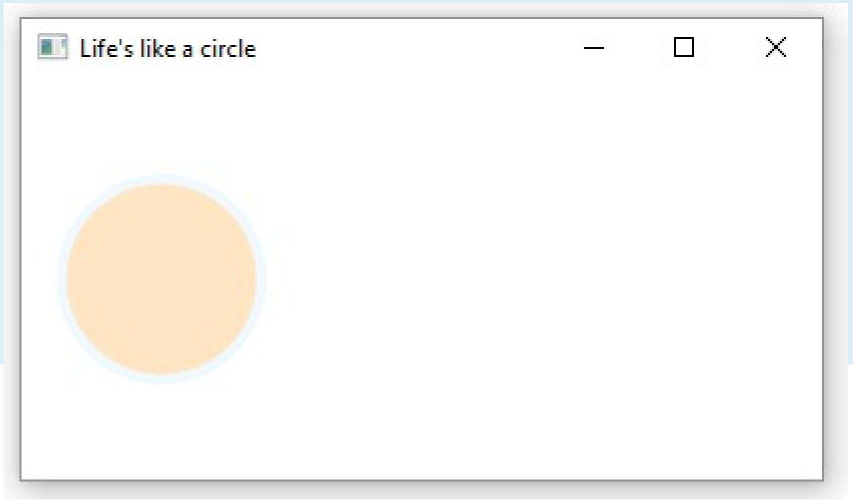
Circle

```
public void start(Stage stg1) throws Exception {
      Circle c = new Circle();
      c.setCenterX(200);
      c.setCenterY(100);
      c.setRadius(50);
      c.setStroke(Color.ALICEBLUE);
      c.setStrokeWidth(5);
      c.setFill(Color.BISQUE);

      Pane pn = new Pane();
      pn.getChildren().add(c);
      Scene scn = new Scene(pn, 400, 200);
      stg1.setTitle("Life's like a circle");
      stg1.setScene(scn);
      stg1.show();
}
```



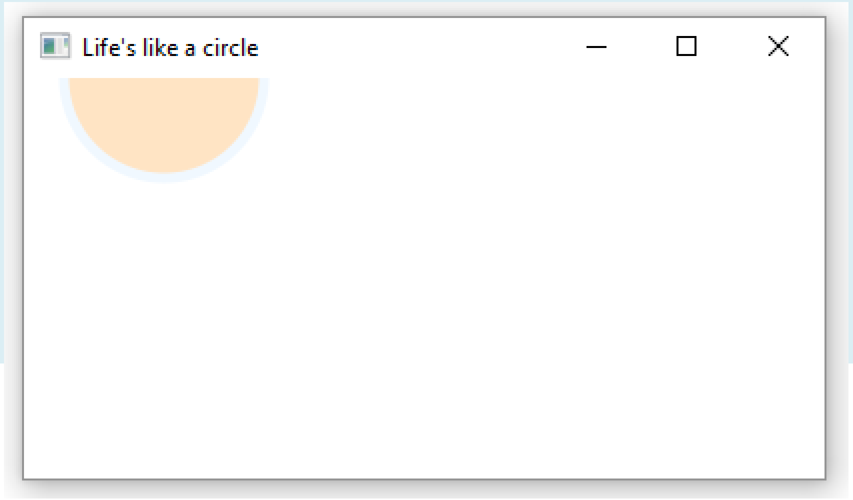
Circle

```
public void start(Stage stg1) throws Exception {
      Circle c = new Circle();
      c.setCenterX(70);
      c.setCenterY(100);
      c.setRadius(50);
      c.setStroke(Color.ALICEBLUE);
      c.setStrokeWidth(5);
      c.setFill(Color.BISQUE);

      Pane pn = new Pane();
      pn.getChildren().add(c);
      Scene scn = new Scene(pn, 400, 200);
      stg1.setTitle("Life's like a circle");
      stg1.setScene(scn);
      stg1.show();
}
```



Circle

```
public void start(Stage stg1) throws Exception {
        Circle c = new Circle();
        c.setCenterX(70);
        c.setCenterY(0);
        c.setRadius(50);
        c.setStroke(Color.ALICEBLUE);
        c.setStrokeWidth(5);
        c.setFill(Color.BISQUE);

        Pane pn = new Pane();
        pn.getChildren().add(c);
        Scene scn = new Scene(pn, 400, 200);
        stg1.setTitle("Life's like a circle");
        stg1.setScene(scn);
        stg1.show();
}
```



JavaFX provides many other shapes

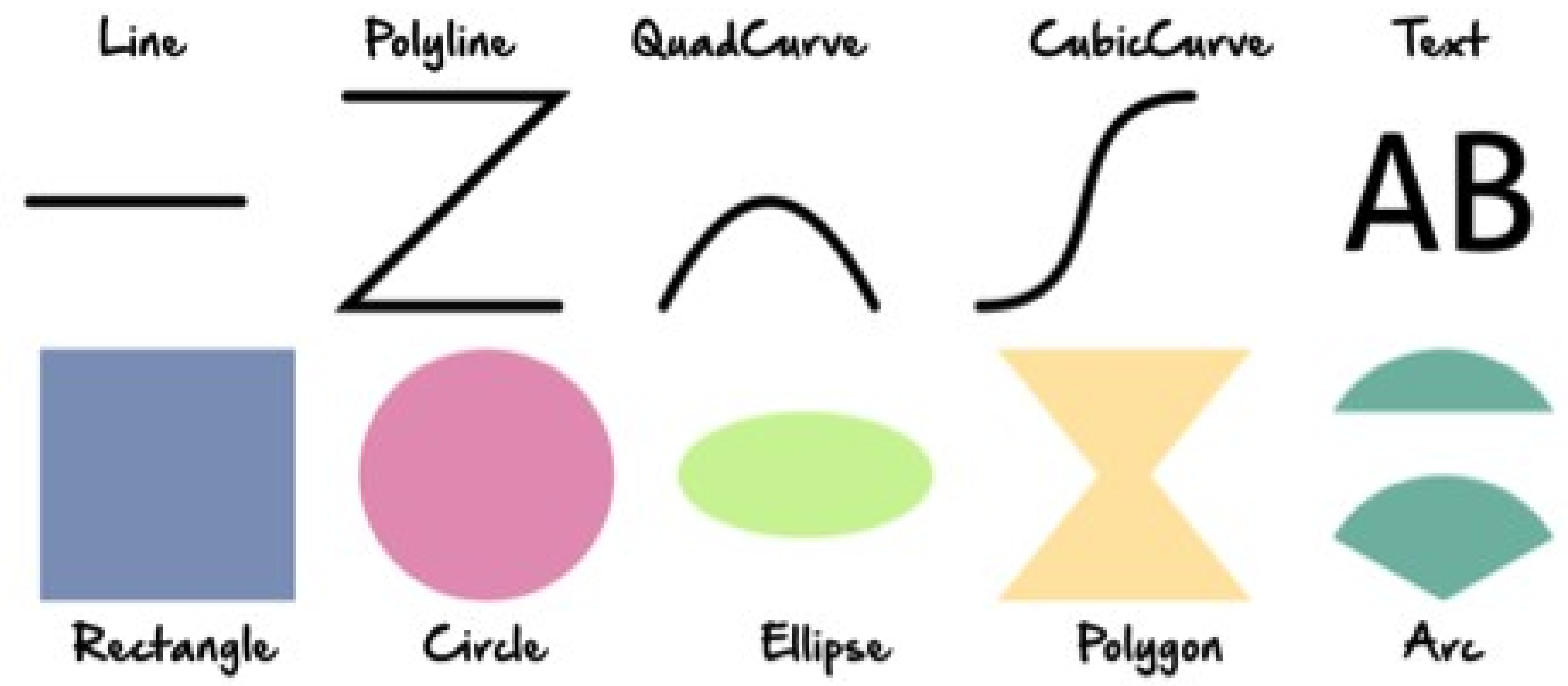
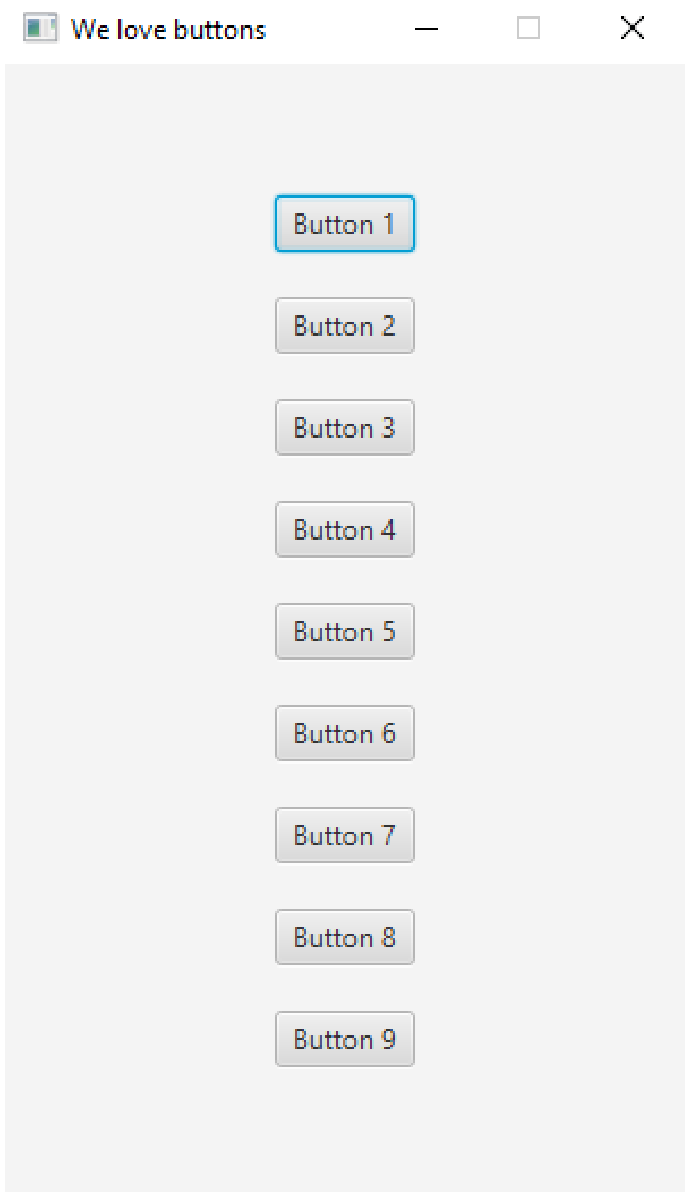


Image copyright: https://dzone.com/refcardz/javafx-8-1?chapter=6

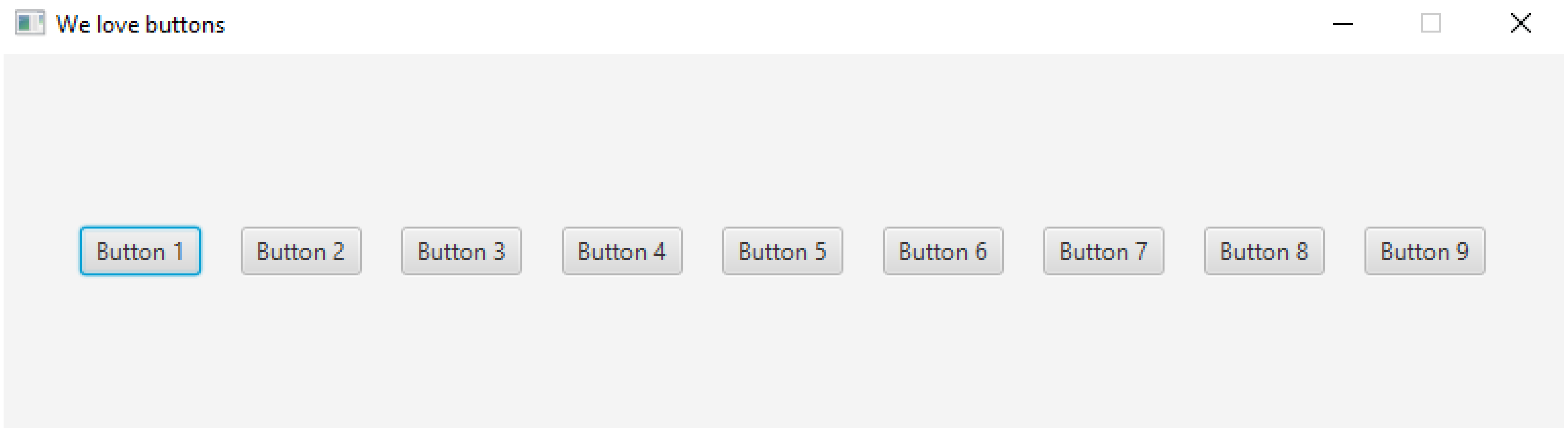
Example: We love buttons



We love buttons

```
     @Override
     public void start(Stage stg1) throws Exception {
          VBox pn = new VBox();
          pn.setAlignment(Pos.CENTER);
          pn.setSpacing(20);
          for(int i = 1; i <= 9; i++)
               pn.getChildren().add(new Button("Button " + 
i));
          Scene scn = new Scene(pn, 300, 500);
          stg1.setTitle("We love buttons");
          stg1.setScene(scn);
          stg1.setResizable(false);
          stg1.show();
     }
```

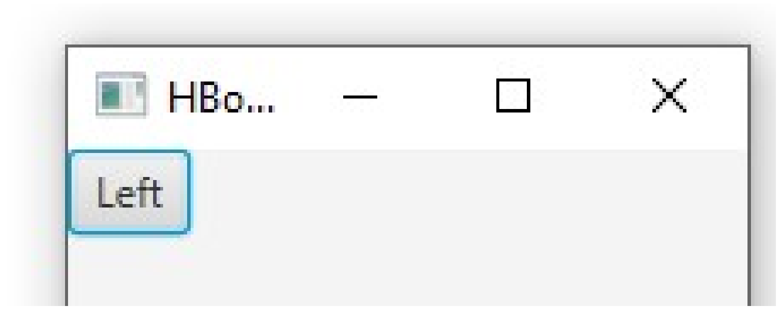
Example: We love horizontal buttons

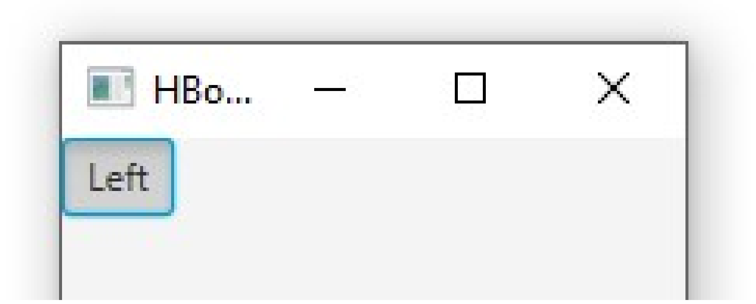


We love horizontal buttons

```
@Override
     public void start(Stage stg1) throws Exception {
          HBox pn = new HBox();
          pn.setAlignment(Pos.CENTER);
          pn.setSpacing(20);
          for(int i = 1; i <= 9; i++)
               pn.getChildren().add(new Button("Button " +
i));
          Scene scn = new Scene(pn, 800, 200);
          stg1.setTitle("We love buttons");
          stg1.setScene(scn);
          stg1.setResizable(false);
          stg1.show();
     }
```

Toggle Buttons

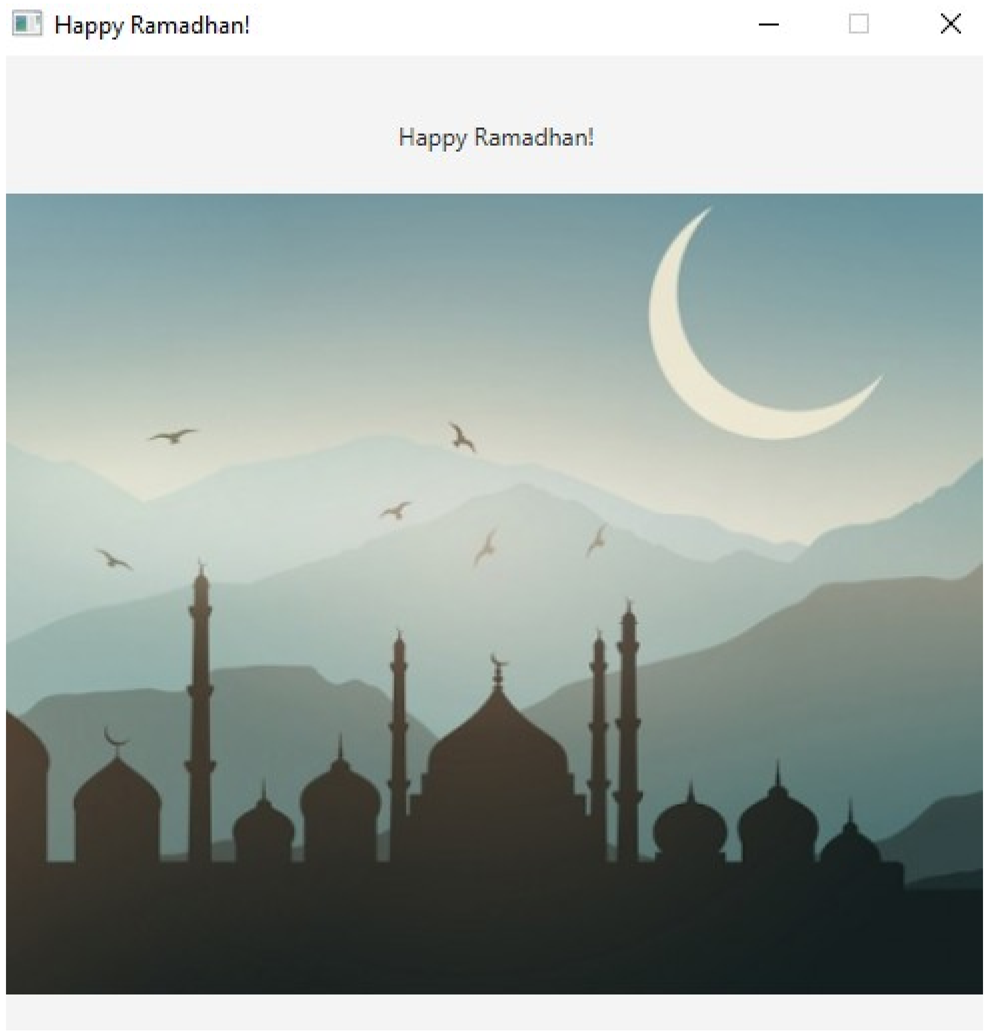




Toggle Buttons

```
@Override
   public void start(Stage primaryStage) throws Exception {
       primaryStage.setTitle("HBox Experiment 1");
       ToggleButton toggleButton1 = new ToggleButton("Left");
       HBox hbox = new HBox(toggleButton1);
       Scene scene = new Scene(hbox, 200, 100);
       primaryStage.setScene(scene);
       primaryStage.show();
   }
```

JavaFX Labels and Images

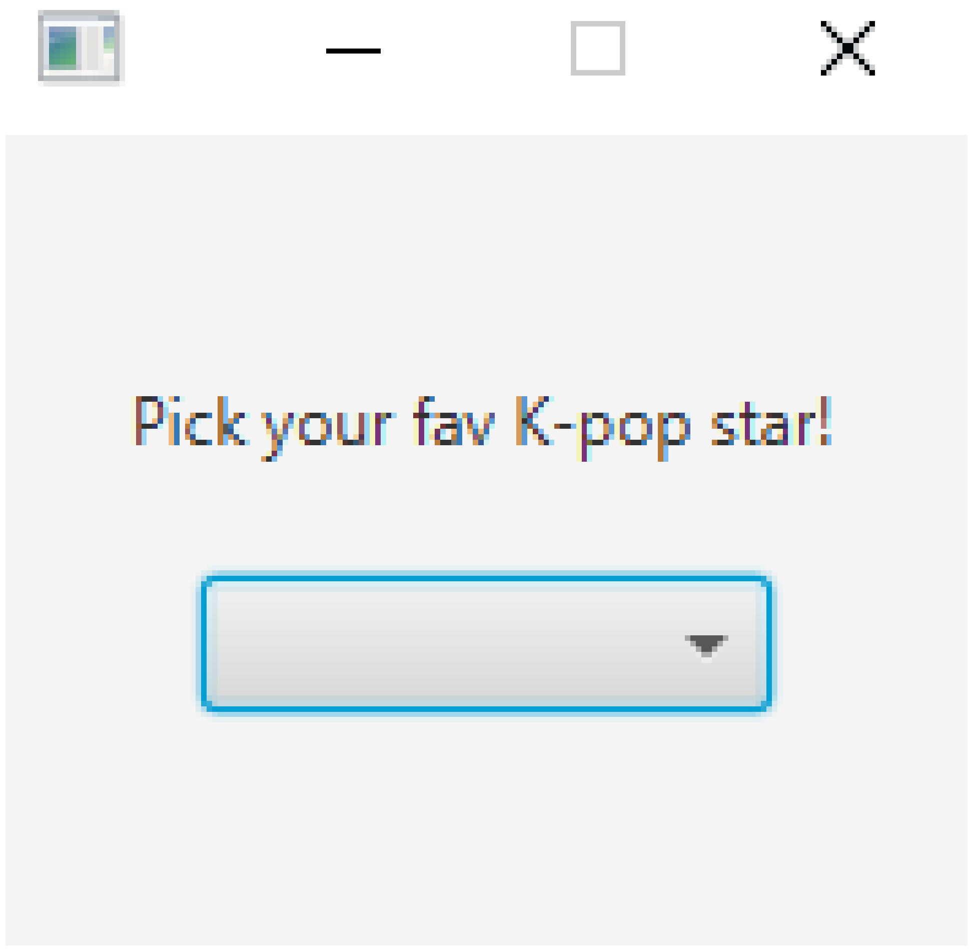


Happy Ramadhan!

JavaFX Labels and Images

```
public void start(Stage stg1) throws Exception {
      VBox pn = new VBox();
      pn.setAlignment(Pos.CENTER);
      pn.setSpacing(20);
      pn.getChildren().add(new Label("Happy Ramadhan!"));
      FileInputStream fis = new FileInputStream("pics/ramadhan.png");
      Image img = new Image(fis);
      ImageView iv = new ImageView(img);
      iv.setFitHeight(400);
      iv.setPreserveRatio(true);
      pn.getChildren().add(iv);
      Scene scn = new Scene(pn, 500, 500);
      stg1.setTitle("Happy Ramadhan!");
      stg1.setScene(scn);
      stg1.setResizable(false);
      stg1.show();
}
```

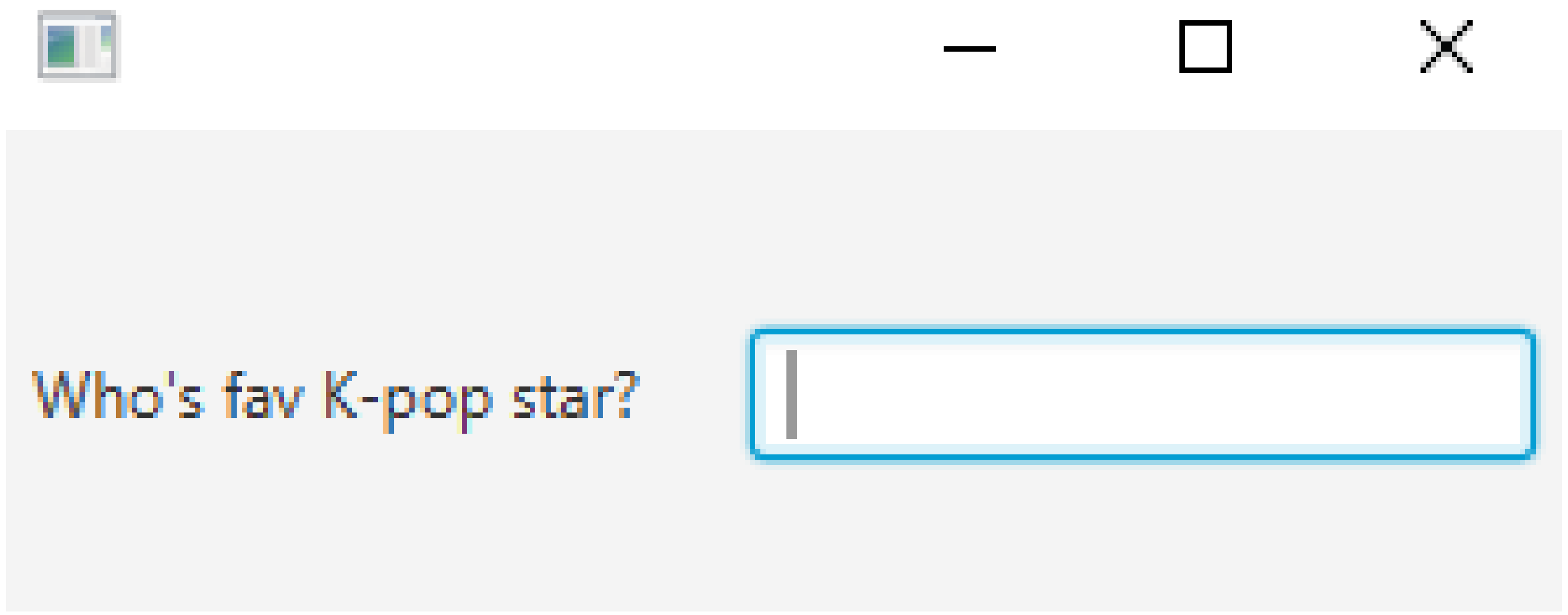
ChoiceBox


Pick your fav K-pop star!

ChoiceBox: Code

```
	public void start(Stage stg1) throws Exception {
		VBox pn = new VBox();
		pn.setAlignment(Pos.CENTER);
		pn.setSpacing(20);
		pn.getChildren().add(new Label("Pick your fav K-pop 
star!"));
		String[] stars = {"Joo Ko-Wee", "Park Bo-Wow"};
		ChoiceBox cb = new 
ChoiceBox(FXCollections.observableArrayList(stars));
		pn.getChildren().add(cb);
		Scene scn = new Scene(pn, 180, 150);
		stg1.setScene(scn);
		stg1.setResizable(false);
		stg1.show();
	}
```

TextField

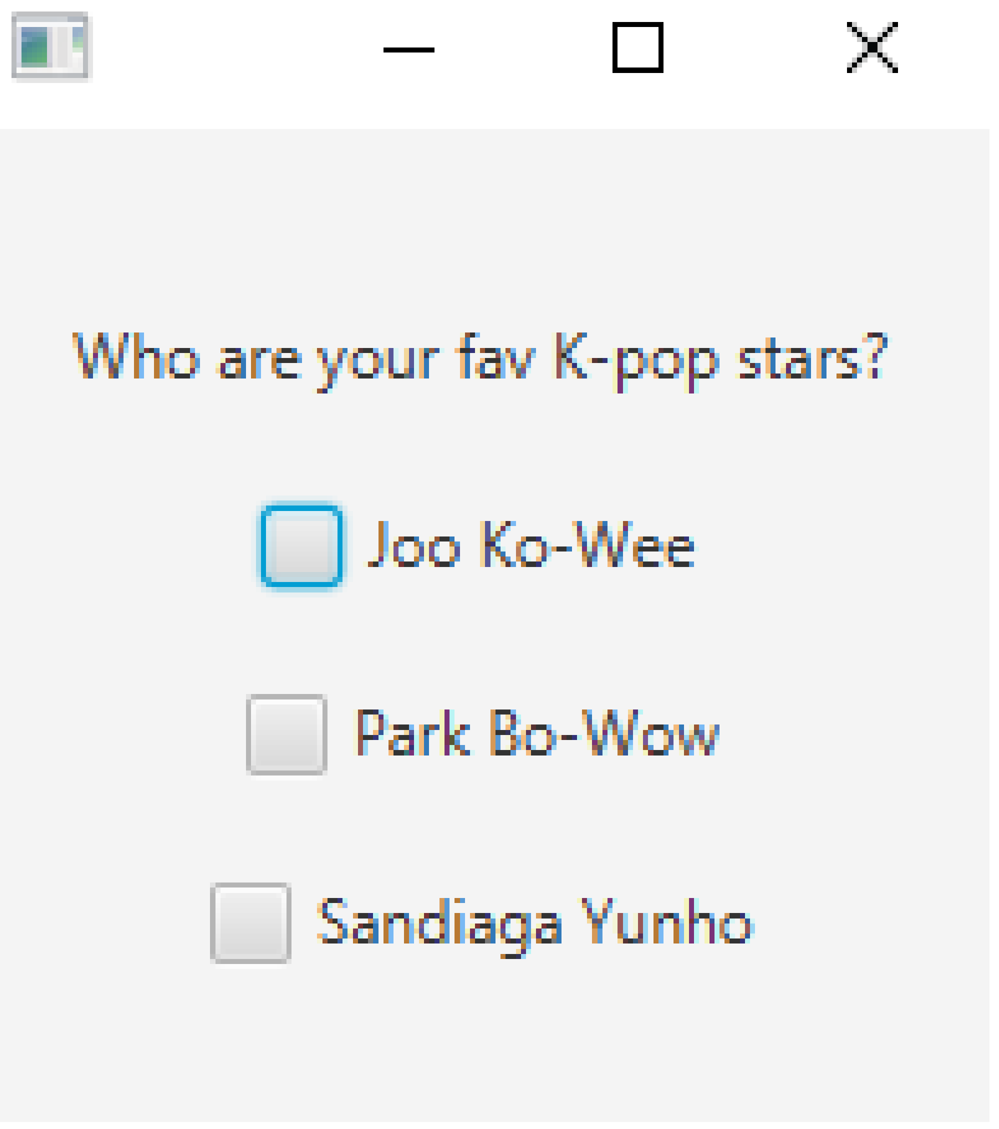


Who's fav K-pop star?

TextField: Code

```
     public void start(Stage stg1) throws Exception {
          HBox pn = new HBox();
          pn.setAlignment(Pos.CENTER);
          pn.setSpacing(20);
          pn.getChildren().add(new Label("Who's fav K-pop 
star?"));
          TextField tf = new TextField();
          tf.setPrefWidth(150);
          pn.getChildren().add(tf);
          Scene scn = new Scene(pn, 300, 100);
          stg1.setScene(scn);
          stg1.show();
     }
```

CheckBox


Who are your fav K-pop stars?

- [ ] Joo Ko-Wee
- [ ] Park Bo-Wow
- [ ] Sandiaga Yunho

CheckBox: Code

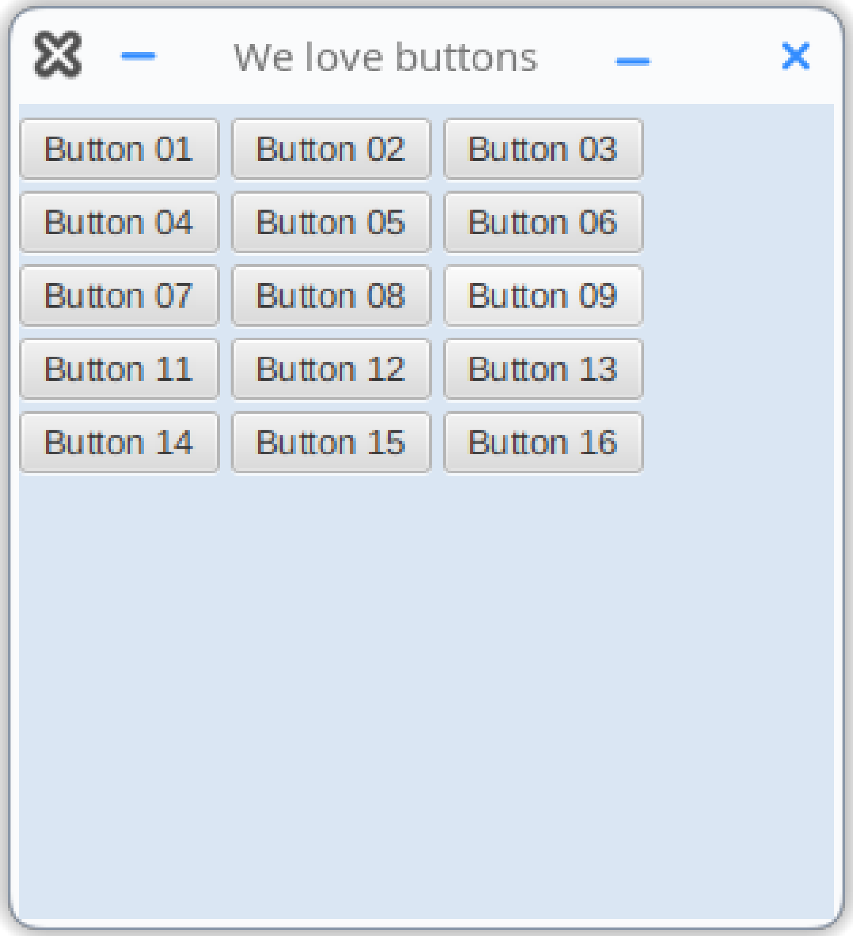
```
	public void start(Stage stg1) throws Exception {
		VBox pn = new VBox();
		pn.setAlignment(Pos.CENTER);
		pn.setSpacing(20);
		pn.getChildren().add(new Label("Who are your fav K-pop 
stars?"));
		CheckBox cb1 = new CheckBox("Joo Ko-Wee");
		CheckBox cb2 = new CheckBox("Park Bo-Wow");
		CheckBox cb3 = new CheckBox("Sandiaga Yunho");
		pn.getChildren().add(cb1);
		pn.getChildren().add(cb2);
		pn.getChildren().add(cb3);
		Scene scn = new Scene(pn, 200, 200);
		stg1.setScene(scn);
		stg1.show();
	}
```

More Examples

- https://docs.oracle.com/javase/8/javafx/get-started-tutorial/get_start_apps.htm

Exercise: We love more buttons!

1) Design a javafx app using FlowPane that shows 16 buttons!
2) The same app with no 1., but use FXML and the SceneBuilder



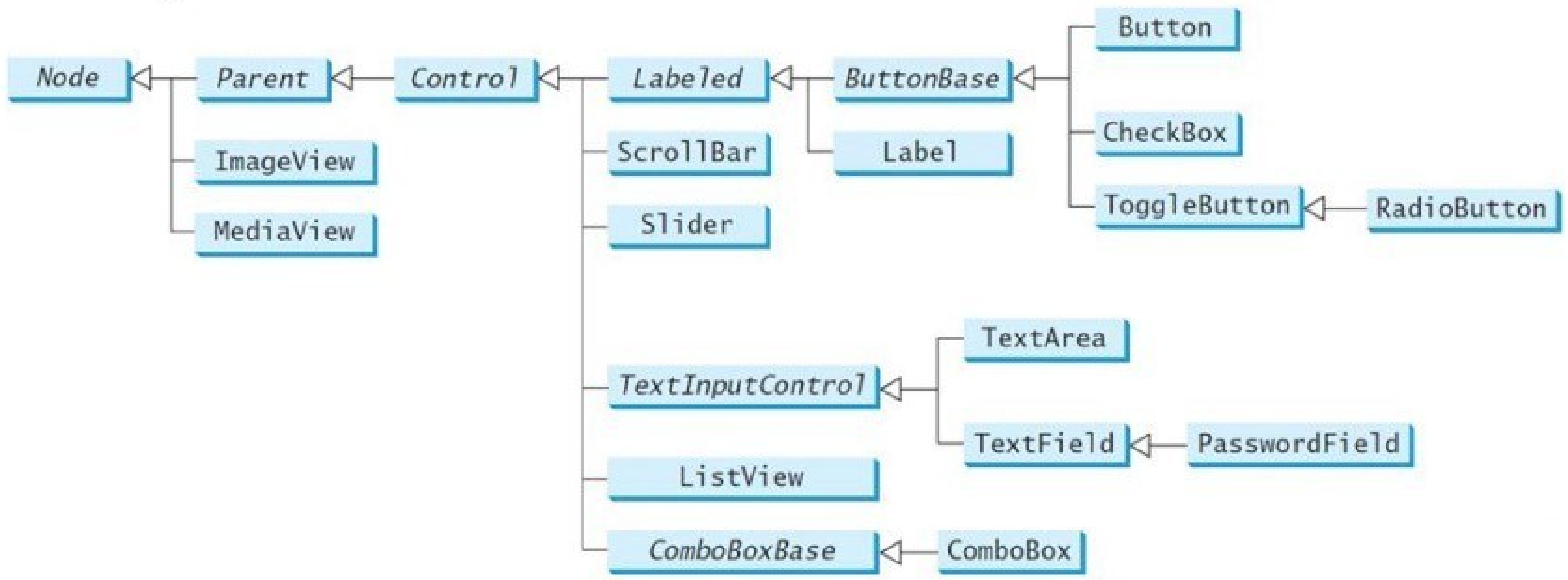
Event Programming

Event Programming

- Procedural programming is executed in procedural order
  - In event-driven programming, code is executed upon activation of events
- Operating Systems constantly monitor events (Ex: keystrokes, mouse clicks, etc…), and, the OS:
  - sorts out these events
  - reports them to the appropriate programs
- How? For each control (button, combo box, etc.)
  - define an event handler
  - construct an instance of event handler
  - tell the control who its event handler is
- Event Handler?
  - code with response to event
  - a.k.a. event listener

Java’s Event Handling

- An event source is a GUI control
  - JavaFX: Button, ChoiceBox, etc.
- Different types of sources:
  - can detect different types of events
  - can register different types of listeners (handlers)



Event Creation

```
public class HelloWorld extends Application { public
    static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) { // entry
        point primaryStage.setTitle("Hello World!");
        Button btn = new Button(“Say Hello World”);

        btn.setOnAction(new HelloEvent()); StackPane pane
        = new StackPane(); pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage
        primaryStage.show();
    }
}

.........
```

- When the user interacts with a control (source):
  - an event object is constructed
    - Contain information about the event
    - Like what?
      - location of mouse click
      - event source that was interacted with, etc.
  - the event object is sent to all registered listener objects
  - the listener object (handler) responds as you defined it to

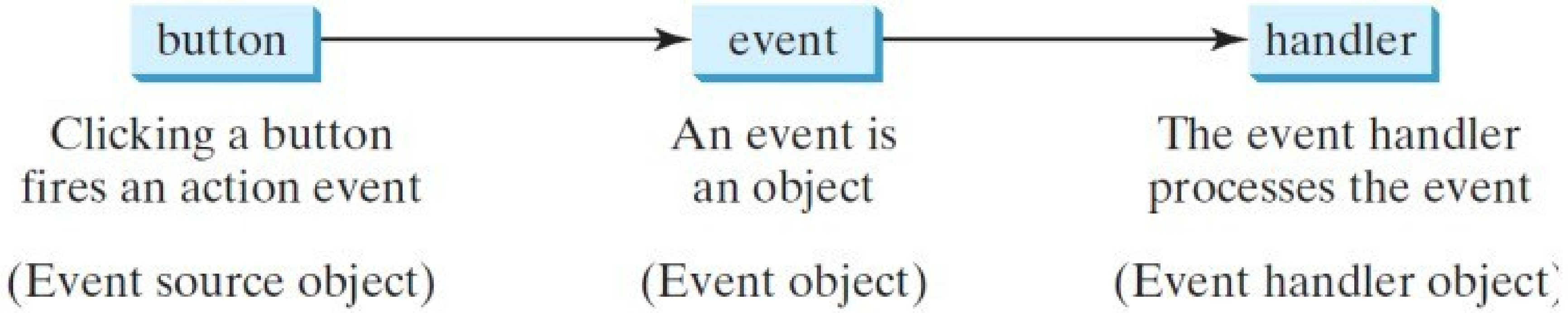
Event Listeners

```
public class HelloWorld extends Application { public static
    void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) { // entry point
        primaryStage.setTitle("Hello World!");
        Button btn = new Button("Say Hello World");

        btn.setOnAction(new HelloEvent()); StackPane pane = new
        StackPane(); pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage
        primaryStage.show();
    }
}
class HelloEvent implements EventHandler<ActionEvent>{
     @Override
     public void handle(ActionEvent event) {
     System.out.println("Hello World!");
     }
}
```

- Event listeners (event handler)
  - Defined by you, the application programmer
    - you customize the response
    - How?
      - Inheritance & Polymorphism
  - You define your own listener class
    - implement the appropriate interface
    - define responses in all necessary methods

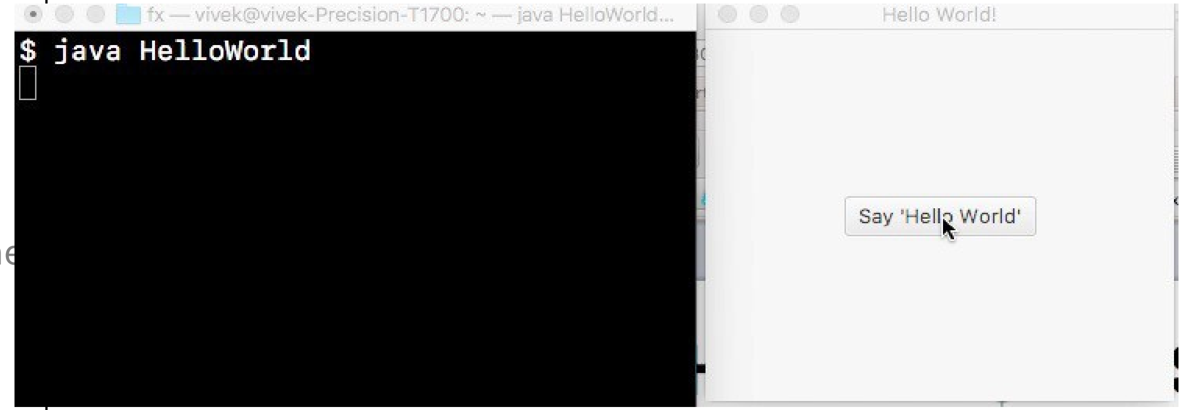
Summary: How to Handle GUI Events

- Source object: button
    - An event is generated by external user actions such as mouse movements, mouse clicks, or keystrokes
- An event can be defined as a type of signal to the program that something has happened
- Listener object contains a method for processing the event.



Working of Our Hello World GUI

```
public class HelloWorld extends Application { public static
    void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) { // entry point
        primaryStage.setTitle("Hello World!");
        Button btn = new Button(“Say Hello World”);

        btn.setOnAction(new HelloEvent()); StackPane pane = ne
        StackPane(); pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage
        primaryStage.show();
    }
}
class HelloEvent implements EventHandler<ActionEvent> {
     @Override
     public void handle(ActionEvent event {
     System.out.println("Hello World!");
     }
}
```



```
$ java HelloWorld
```

Productivity in Event Programming

```
public class HelloWorld extends Application { public
    static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) { // entry
        point primaryStage.setTitle("Hello World!");
        Button btn = new Button(“Say Hello World”);

        btn.setOnAction(new HelloEvent()); StackPane pane
        = new StackPane(); pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage
        primaryStage.show();
    }
}
class HelloEvent implements EventHandler<ActionEvent> {
     @Override
     public void handle(ActionEvent event) {
          System.out.println("Hello World!");
     }
     }
```

- Can we write this code in much better way?

Productivity in Event Programming (1/3)

```
public class HelloWorld extends Application { public
    static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) { // entry
        point primaryStage.setTitle("Hello World!");
        Button btn = new Button(“Say Hello World”);

        btn.setOnAction(new HelloEvent()); StackPane pane
        = new StackPane(); pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage
        primaryStage.show();
    }
    class HelloEvent implements
    EventHandler<ActionEvent> {
         @Override
         public void handle(ActionEvent event)
             { System.out.println("Hello World!");
         }
    }
}
```

- Using **inner** classes for creating listener objects

Productivity in Event Programming (2/3)

```
public class HelloWorld extends Application { public static
    void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) { // entry point
        primaryStage.setTitle("Hello World!");
        Button btn = new Button("Say Hello World");

        btn.setOnAction(new EventHandler<ActionEvent>({ \

            @Override public void handle(ActionEvent event){

                System.out.println("Hello World!");
            }
        });
        StackPane pane = new StackPane();
        pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage
        primaryStage.show();
    }
}
```

Using **anonymous** inner classes for creating listener objects

- It combines declaring an inner class and creating an instance of the class in one step
- An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause
  - An anonymous inner class must implement all the abstract methods in the superclass or in the interface
- An anonymous inner class always uses the no-arg constructor from its superclass to create an instance

Productivity in Event Programming (3/3)

```
public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) { // entry point
        primaryStage.setTitle("Hello World!");
        Button btn = new Button("Say Hello World");

        btn.setOnAction(e -> {
            System.out.println("Hello World!");
        });
        StackPane pane = new StackPane();
        pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage
        primaryStage.show();
    }
}
```

- **Using lambda expressions to simplify event handling**
  - Lambda expressions can be viewed as an anonymous method with a concise syntax
  - The statements in the lambda expression is all for that method
  - If it contains multiple methods, the compiler will not be able to compile the lambda expression
  - So, for the compiler to understand lambda expressions, the interface must contain exactly one method

Collection Elements (1/4)

```
public class Test {
    Map<String, Integer> items =
                new HashMap<String, Integer>();

    public void addElements() { ..... }

    public void print() {
        for(Map.Entry<String, Integer> entry
                     : items.entrySet())
            { System.out.println(entry.getKey()
                     + “, “ +
                     entry.getValue());
        }
    }

}
```

- Till now we know only this way to iterate over a collection (e.g., Map)
- Drawback
  - Slightly inconvenient coding

Collection Elements (2/4)

```
public class Test {
    Map<String, Integer> items =
                new HashMap<String, Integer>();

    public void addElements() { ..... }

    public void print() {
        items.forEach( (k, v) -> {
            System.out.println(k + “, “ + v);
        });
    }

}
```

- Java 8 introduces **forEach** statement to ease iterating over the collection elements
- With **lambda expressions** in Java 8 this code becomes very compact now!

Collection Elements (3/4)

```
public class Test {
    Map<String, Integer> items =
                 new HashMap<String, Integer>();

    public void addElements() { ..... }

    public void print() {
        items.forEach( (k, v) -> {
            if("ABC".equals(k))
                { System.out.println("Hello
                ABC!");
            }
            System.out.println(k + ", " + v);
        });
    }

}
```

- You can do some more stuff inside that lambda function!

Collection Elements (4/4)

```
public class Test {
    Map<String, Integer> items =
                new HashMap<String, Integer>();

    public void addElements() { ..... } public

    void print() {
        items.forEach( (String k, Integer v) -> {
            if(“ABC”.equals(k))
                { System.out.println(“Hello ABC!”);
            }
            System.out.println(k + “, “ + v);
        });
    }

}
```

- You can even declare type of variables in lambda function

Productivity in Event Programming (3/3)

```
public class HelloWorld extends Application { public
    static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) { // entry
        point primaryStage.setTitle("Hello World!");
        Button btn = new Button("Say Hello World");

        btn.setOnAction(e -> { System.out.println("Hello
            World!");
        });
        StackPane pane = new StackPane();
        pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 200, 50);
        // Place the scene in the stage
        primaryStage.setScene(scene);
        // Display the stage primaryStage.show();
    }
}
```

- Using Java **lambda** expressions to simplify event handling
  - Lambda expressions can be viewed as an anonymous method with a concise syntax
  - The statements in the lambda expression is all for that method
  - If it contains multiple methods, the compiler will not be able to compile the lambda expression
  - So, for the compiler to understand lambda expressions, the interface must contain exactly one method