# Java Class 12

OKAY, JUST SHOW ME HOW TO PROGRAM SO I CAN HELP OUT ON YOUR PROJECT.
YOU'RE GOING TO BUILD A "G.U.I." USING OBJECT-ORIENTED DEVELOPMENT TOOLS...
G.U.I. IS PRONOUNCED "GOOEY."
I USED MY GUN OBJECT TO BLAST THE BUG OBJECT IN THE HALL OBJECT!!
NOTICE HOW GOOEY IT IS.
S. Adams
E-mail: SCOTTADAMS@AOL.COM
© 1996 United Feature Syndicate, Inc.(NYC)

## Arcs

In JavaFX, an arc is defined as a portion of an ellipse

Like an ellipse, the first four parameters to the Arc constructor specify the center point (x and y) as well as the radii along the horizontal and vertical

Two additional parameters specify the portion of the ellipse that define the arc

## Arcs

The Arc constructor:

```
Arc(centerX, centerY, radiusX, radiusY, startAngle, arcLength)
```

The *start angle* is where the arc begins relative to the horizontal

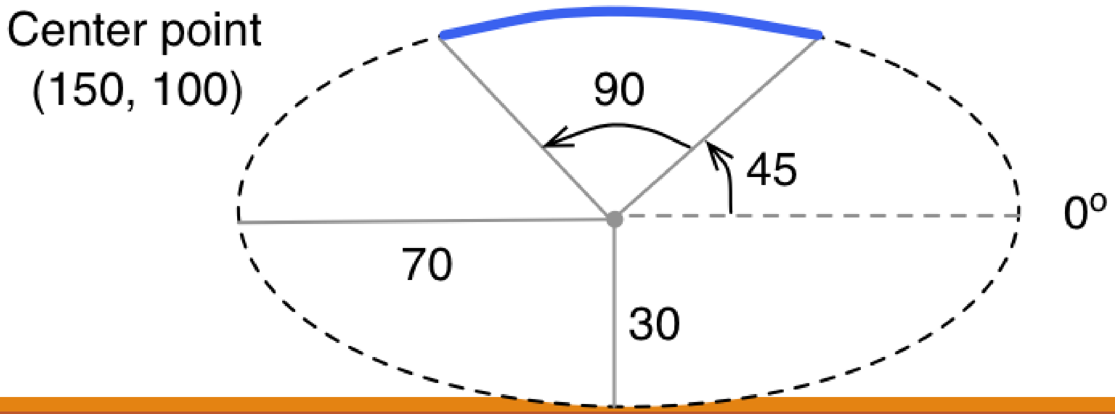
The *arc length* is the angle that defines how big the arc is

Both angles are specified in degrees

# Arcs

An arc whose underlying ellipse is centered at (150, 100), a horizontal radius of 70 and a vertical radius of 30, a start angle of 45 and a arc length of 90:

```
Arc myArc = new Arc(150, 100, 70, 30, 45, 90);
```



## Arcs

An arc also has an *arc type*:

| | |
|---|---|
| **ArcType.OPEN** | The curve along the ellipse edge |
| **ArcType.CHORD** | End points are connected by a straight line |
| **ArcType.ROUND** | End points are connected to the center point of the ellipse, forming a rounded “pie” piece |

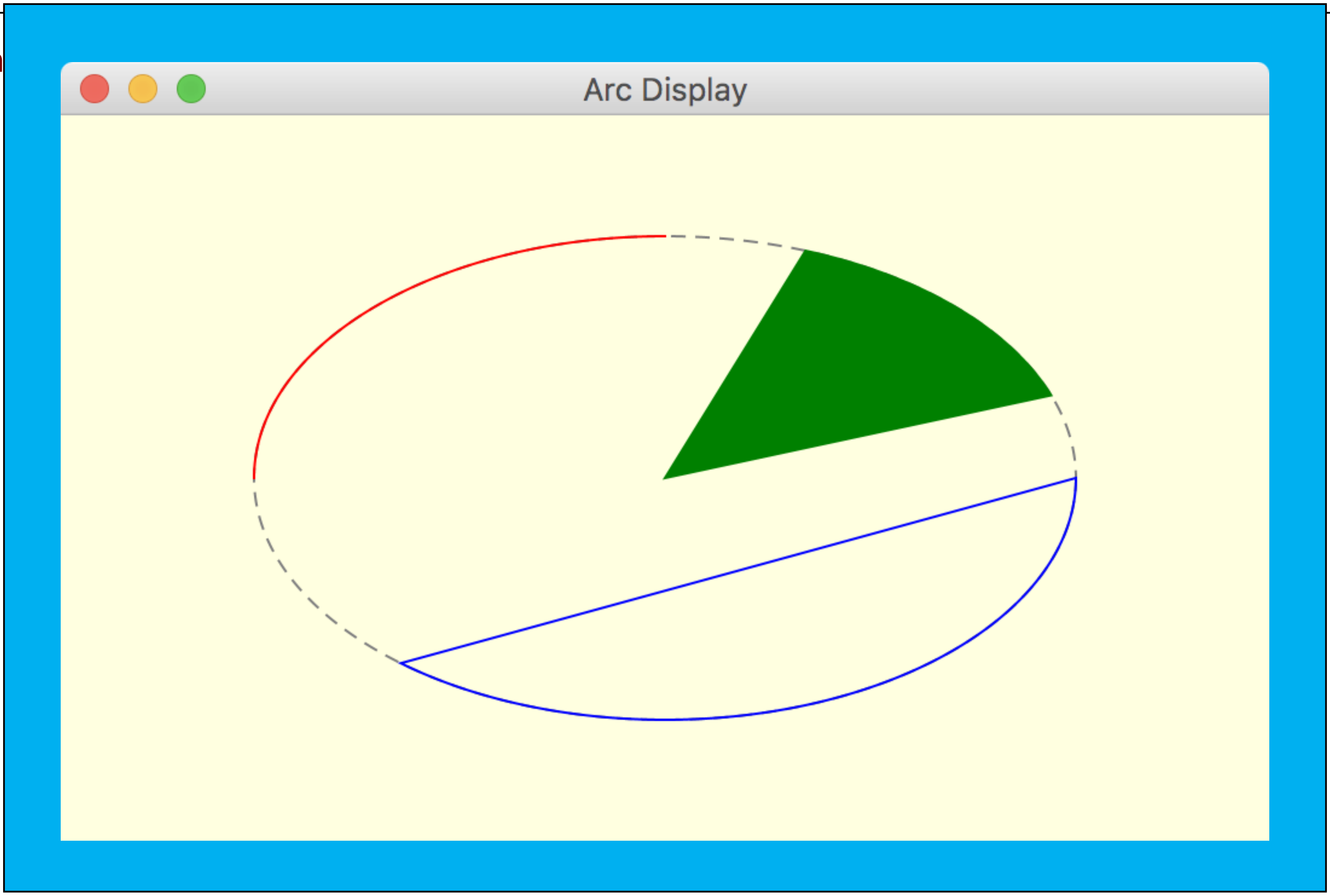
```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.paint.Color;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcType;
import javafx.scene.shape.Ellipse;
import javafx.stage.Stage;

//************************************************************************
//  ArcDisplay.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the JavaFX Arc class.
//************************************************************************

public class ArcDisplay extends Application
{
    //--------------------------------------------------------------------
    //  Draws three arcs based on the same underlying ellipse.
    //--------------------------------------------------------------------
    public void start(Stage primaryStage)
    {
        Ellipse backgroundEllipse = new Ellipse(250, 150, 170, 100);
        backgroundEllipse.setFill(null);
        backgroundEllipse.setStroke(Color.GRAY);
        backgroundEllipse.getStrokeDashArray().addAll(5.0, 5.0);
```

continue

continue

```
        Arc arc1 = new Arc(250, 150, 170, 100, 90, 90);
        arc1.setType(ArcType.OPEN);
        arc1.setStroke(Color.RED);
        arc1.setFill(null);

        Arc arc2 = new Arc(250, 150, 170, 100, 20, 50);
        arc2.setType(ArcType.ROUND);
        arc2.setStroke(Color.GREEN);
        arc2.setFill(Color.GREEN);

        Arc arc3 = new Arc(250, 150, 170, 100, 230, 130);
        arc3.setType(ArcType.CHORD);
        arc3.setStroke(Color.BLUE);
        arc3.setFill(null);

        Group root = new Group(backgroundEllipse, arc1, arc2, arc3);
        Scene scene = new Scene(root, 500, 300, Color.LIGHTYELLOW);

        primaryStage.setTitle("Arc Display");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

## contin



```
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

## More on Arcs

The start angle or the arc length could have been  specified using negative values.  If negative, the angle is measured clockwise instead of counterclockwise.

Here's an alternative way to specify the red open arc from the example that the previous code produced.

```
Arc arc1 = new Arc(250, 1 50, 1 70, 100, -1 80, -90);
```

## Images

The JavaFX `Image` class is used to load an image from a file or URL

Supported formats:  jpeg, gif, and png

To display an image, use an `ImageView` object

An `Image` object cannot be added to a container directly

```java
import javafx.application.Application;
import javafx.geometry.Rectangle2D;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

//************************************************************************
//   ImageDisplay.java       Author: Lewis/Loftus
//
//   Demonstrates a the use of Image and ImageView objects.
//************************************************************************

public class ImageDisplay extends Application
{
    //--------------------------------------------------------------------
    //  Displays an image centered in a window.
    //--------------------------------------------------------------------
    public void start(Stage primaryStage)
    {
        Image img = new Image("gull.jpg");
        ImageView imgView = new ImageView(img);

        StackPane pane = new StackPane(imgView);
        pane.setStyle("-fx-background-color: cornsilk");
```

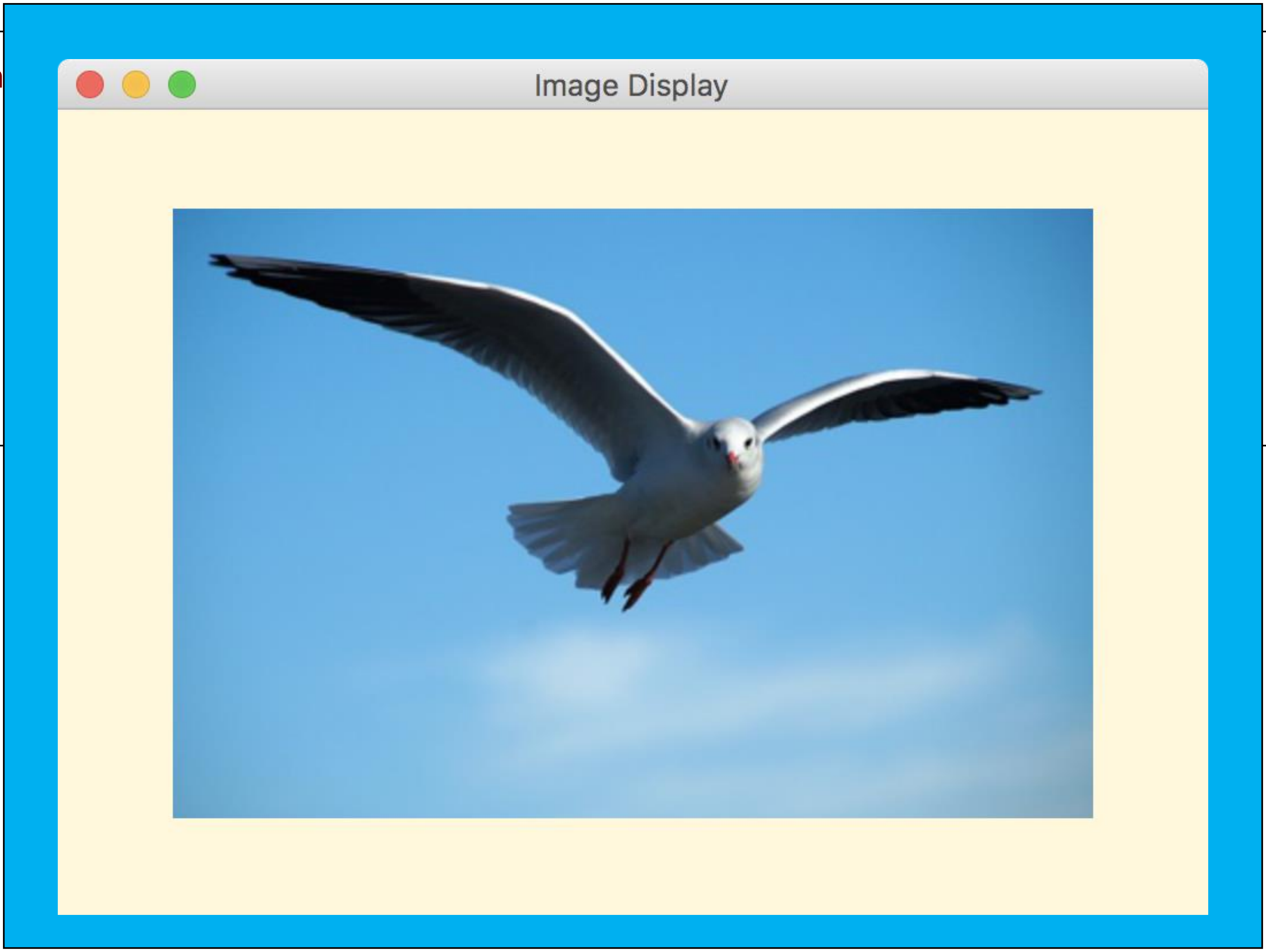
continue

continue

```
        Scene scene = new Scene(pane, 500, 350);

        primaryStage.setTitle("Image Display");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

contin

```
        }
}
```



## Images

The parameter to the `Image` constructor can include a pathname:

```
Image logo = new Image("myPix/smallLogo.png");
```

It can also be a URL:

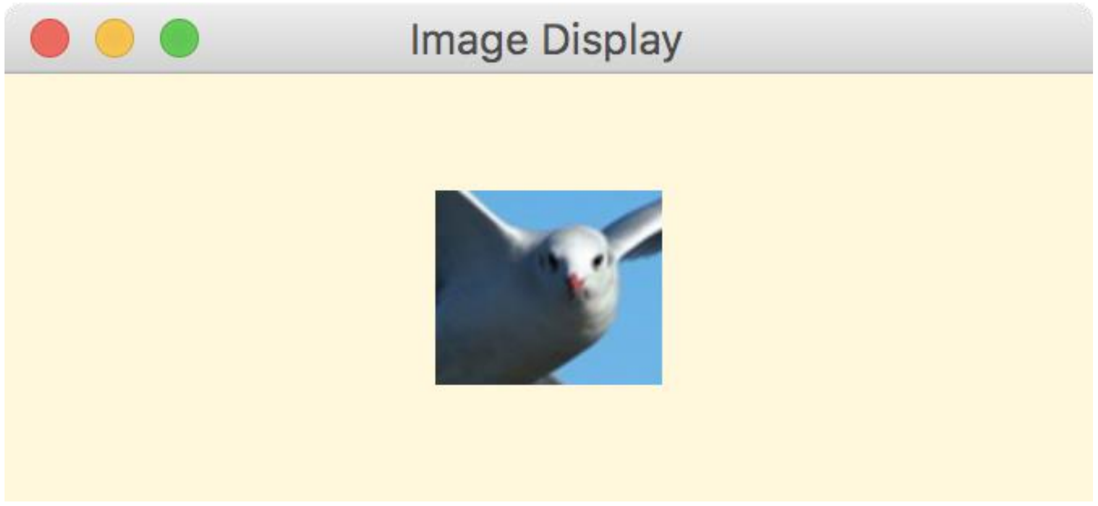
```
Image logo = new("http://example.com/images/bio.jpg");
```

## Viewports

A *viewport* is a rectangular area that restricts the pixels displayed in an `ImageView`

It is defined by a `Rectangle2D` object:

```
imgView.setViewport(new Rectangle2D(200, 80, 70, 60));
```



## Graphical User Interfaces

A Graphical User Interface (GUI) in Java is created with at least three kinds of objects:

- controls, events, and event handlers

A *control* is a screen element that displays information or allows the user to interact with the program:

- labels, buttons, text fields, sliders, etc.

## Graphical User Interfaces

An *event* is an object that represents some activity to which we may want to respond

For example, we may want our program to perform some action when the following occurs:

- a graphical button is pressed
- a slider is dragged
- the mouse is moved
- the mouse is dragged
- the mouse button is clicked
- a keyboard key is pressed

## Graphical User Interfaces

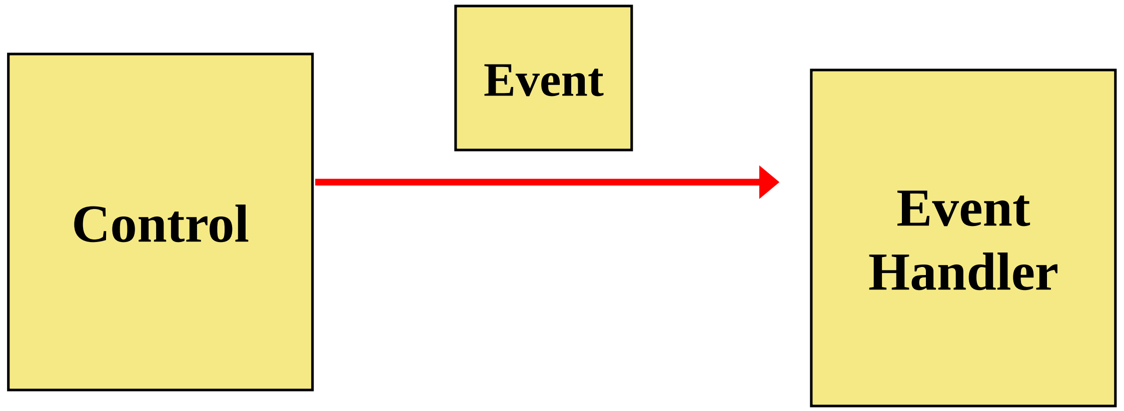
The Java API contains several classes that represent typical events

Controls, such as a button, generate (or fire) an event when it occurs

We set up an *event handler* object to respond to an event when it occurs

We design event handlers to take whatever actions are appropriate when an event occurs

## Graphical User Interfaces



A control object generates an event

A corresponding event handler is designed to respond to the event

When the event occurs, the control calls the appropriate method of the listener, passing an object that describes the event

# Graphical User Interfaces

A JavaFX *button* is defined by the `Button` class

It generates an *action event*

The `PushCounter` example displays a button that increments a counter each time it is pushed

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.text.Text;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

//************************************************************************
//  PushCounter.java       Author: Lewis/Loftus
//
//  Demonstrates JavaFX buttons and event handlers.
//************************************************************************

public class PushCounter extends Application
{
    private int count;
    private Text countText;
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Presents a GUI containing a button and text that displays
   //  how many times the button is pushed.
   //-----------------------------------------------------------------
   public void start(Stage primaryStage)
   {
       count = 0;
       countText = new Text("Pushes: 0");

       Button push = new Button("Push Me!");
       push.setOnAction(this::processButtonPress);

       FlowPane pane = new FlowPane(push, countText);
       pane.setAlignment(Pos.CENTER);
       pane.setHgap(20);
       pane.setStyle("-fx-background-color: cyan");

       Scene scene = new Scene(pane, 300, 100);

       primaryStage.setTitle("Push Counter");
       primaryStage.setScene(scene);
       primaryStage.show();
   }
```

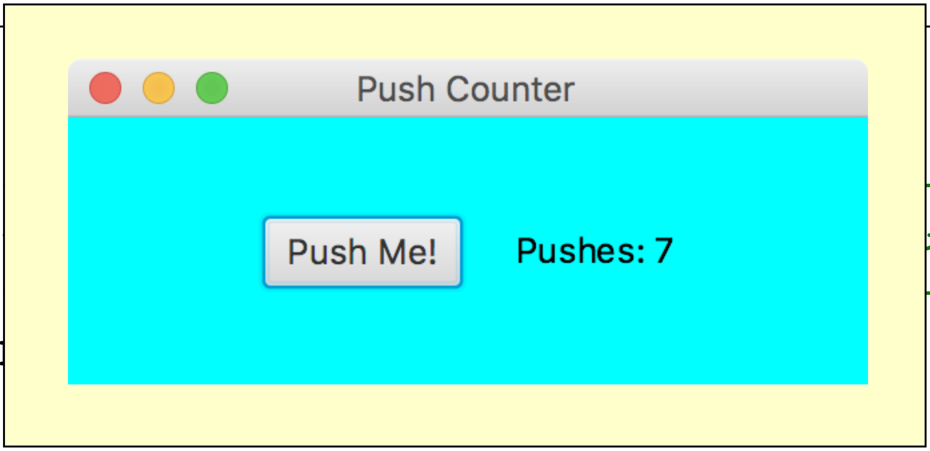
continue

continue

```
   //-----------------------------------------------------------------
   //  Updates the counter and text when the button is pushed.
   //-----------------------------------------------------------------
   public void processButtonPress(ActionEvent event)
   {
      count++;
      countText.setText("Pushes: " + count);
   }
}
```

continue

```
   //----------
   //  Updates 
   //----------
   public void 
   {
      count++;
      countText.setText("Pushes: " + count);
   }
}
```



```
Pushes: 7
```

```
-----------------
ushed.
-----------------
```

## Graphical User Interfaces

A call to the `setOnAction` method sets up the relationship between the button that generates the event and the event handler that responds to it

This example uses a *method reference* (using the `::` operator) to specify the event handler method

The `this` reference indicates that the event handler method is in the same class

So the `PushCounter` class also represents the event handler for this program

## Graphical User Interfaces

The event handler method can be called whatever you want, but must accept an ActionEvent object as a parmeter

In this example, the event handler method increments the counter and updates the text object

The counter and `Text` object are declared at the class level so that both methods can use them

In this example, a `FlowPane` is used as the root node of the scene

A flow pane is another layout pane, which displays its contents horizontally in rows or vertically in columns

A gap of 20 pixels is established between elements on a row using the `setHGap` method

## Determining Event Sources

Recall that you must establish a relationship between controls and the event handlers that respond to events

When appropriate, one event handler object can be used to listen to multiple controls

The source of the event can be determined by using the `getSource` method of the event passed to the event handler

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

//************************************************************************
//   RedOrBlue.java       Author: Lewis/Loftus
//
//   Demonstrates the use of one handler for multiple buttons.
//************************************************************************

public class RedOrBlue extends Application
{
    private Button redButton, blueButton;
    private FlowPane pane;
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Presents a GUI with two buttons that control the color of the
   //  pane background.
   //-----------------------------------------------------------------
   public void start(Stage primaryStage)
   {
      redButton = new Button("Red!");
      redButton.setOnAction(this::processColorButton);

      blueButton = new Button("Blue!");
      blueButton.setOnAction(this::processColorButton);

      pane = new FlowPane(redButton, blueButton);
      pane.setAlignment(Pos.CENTER);
      pane.setHgap(20);
      pane.setStyle("-fx-background-color: white");

      Scene scene = new Scene(pane, 300, 100);

      primaryStage.setTitle("Red or Blue?");
      primaryStage.setScene(scene);
      primaryStage.show();
   }
```

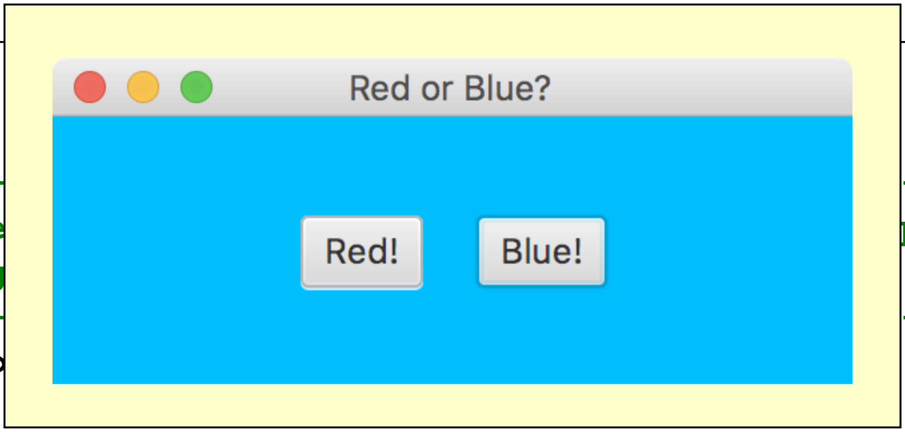
continue

continue

```
    //--------------------------------------------------------------------
    //  Determines which button was pressed and sets the pane color
    //  accordingly.
    //--------------------------------------------------------------------
    public void processColorButton(ActionEvent event)
    {
        if (event.getSource() == redButton)
            pane.setStyle("-fx-background-color: crimson");
        else
            pane.setStyle("-fx-background-color: deepskyblue");
    }
}
```

continue

```
    //-----------                                               -----------------
    //   Determine                                              pane color
    //   according
    //-----------                                               -----------------
    public void p
    {
        if (event.getSource() == redButton)
            pane.setStyle("-fx-background-color: crimson");
        else
            pane.setStyle("-fx-background-color: deepskyblue");
    }
}
```



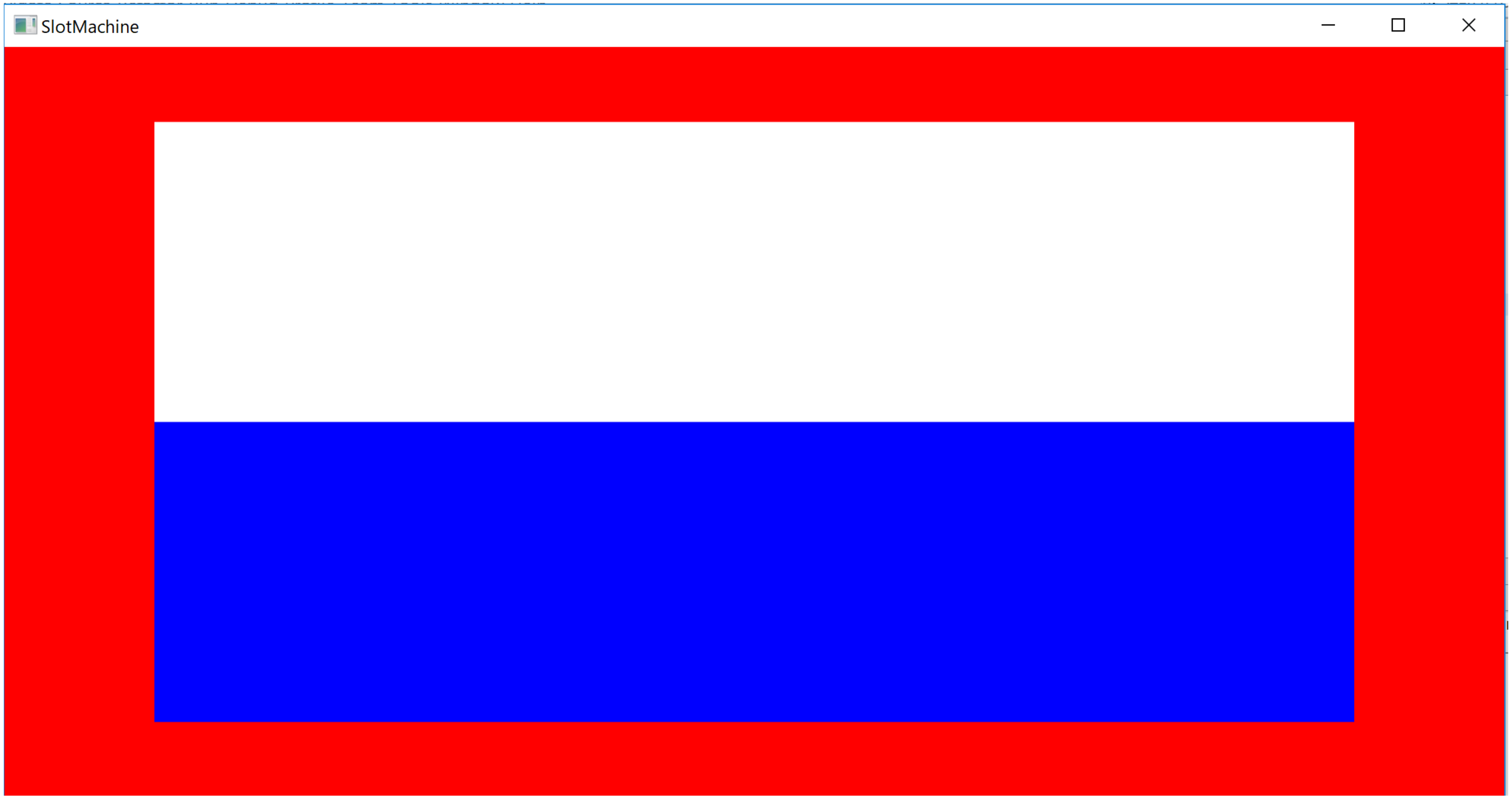
## Individual Exercise – Due 2/25

***Note:  this is an individual assignment!***

Begin Programming Assignment #2

Write an application program that creates a driver class called SlotMachine that creates a SlotMachinePane. SlotMachinePane should display two nested panes (slotsPane and buttonPane) (in different colors) in a vertical box arrangement on a pane called primaryPane.

1. *Set up three variables in SlotMachinePane (hint:  they are all Panes).*
2. *Build your SlotMachinePane constructor that creates the three panes and adds them to SlotMachinePane.*
3. *See following useful code.*


SlotMachine

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class SlotMachine extends Application
{
   public void start(Stage primaryStage)
   {
      SlotMachinePane pane = new SlotMachinePane();
      Scene scene = new Scene(pane, 1000, 500);
      primaryStage.setTitle("SlotMachine");
      primaryStage.setScene(scene);
      primaryStage.show();
   }
     public static void main(String[] args)
   {
      launch(args);
   }
}
```

Driver code

```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;

import java.util.Random;
```

import statements for SlotMachinePane
*(you may not need them all)*

## Assignment for Class 13

Review ArcDisplay, ImageDisplay, PushCounter, RedOrBlue

Read Chapter 5.1, 5.2