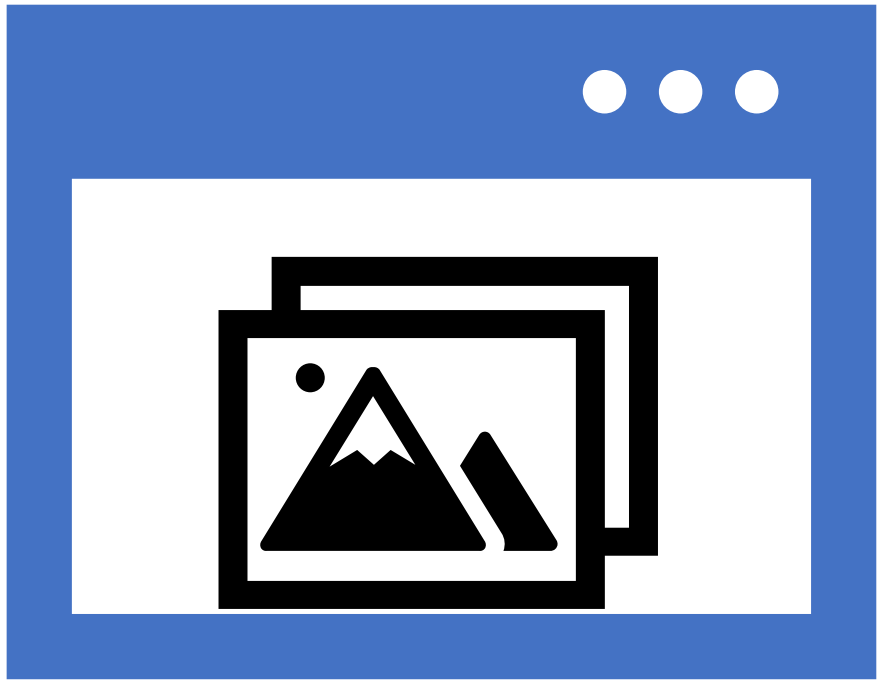
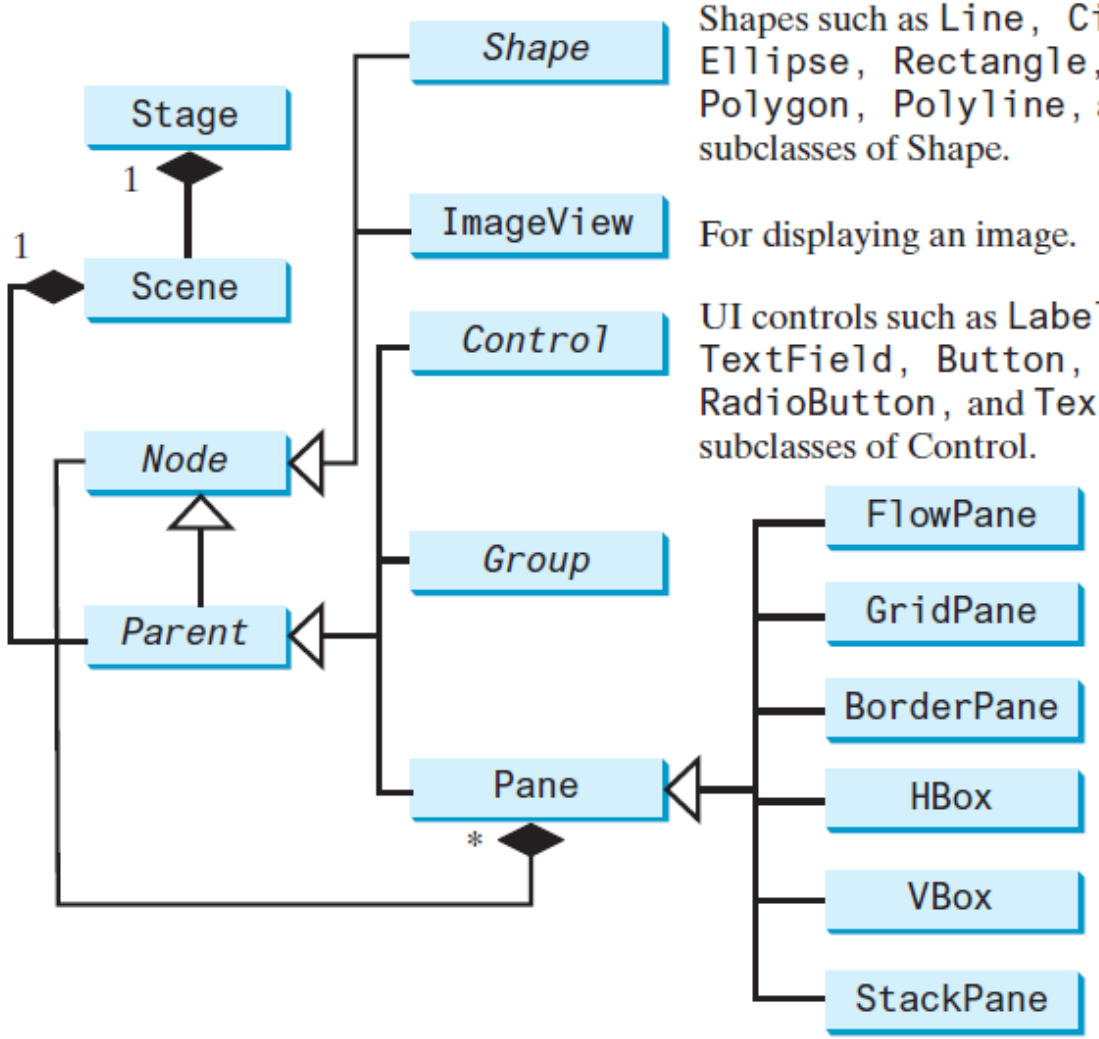


Your graphical application will run in a Window Manager that also reacts to events. For instance you may have an active window and an inactive window on the screen.

Panes, Groups, UI controls, and shapes are subtypes of **Node**.

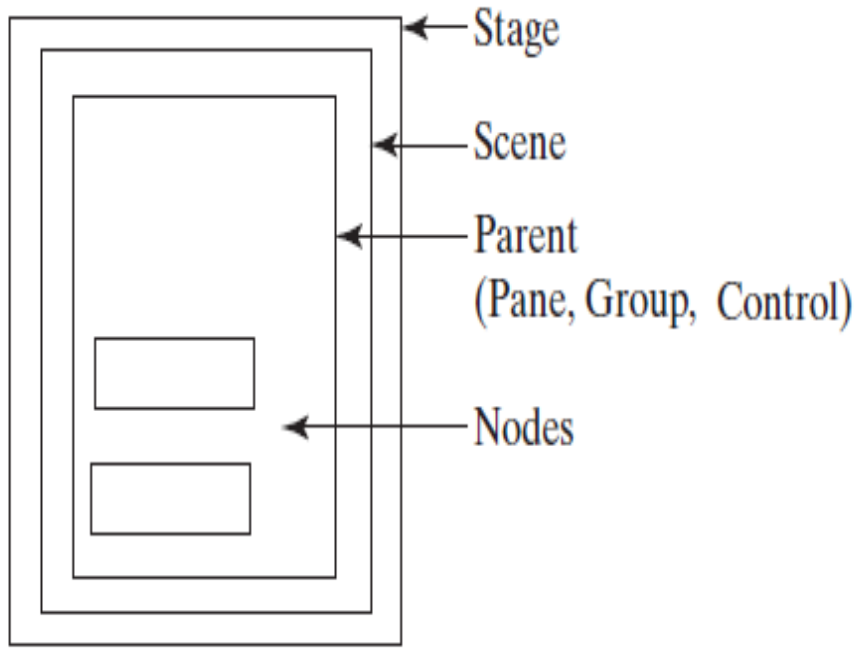


Shapes such as `Line`, `Circle`, `Ellipse`, `Rectangle`, `Arc`, `Polygon`, `Polyline`, and `Text` are subclasses of Shape.

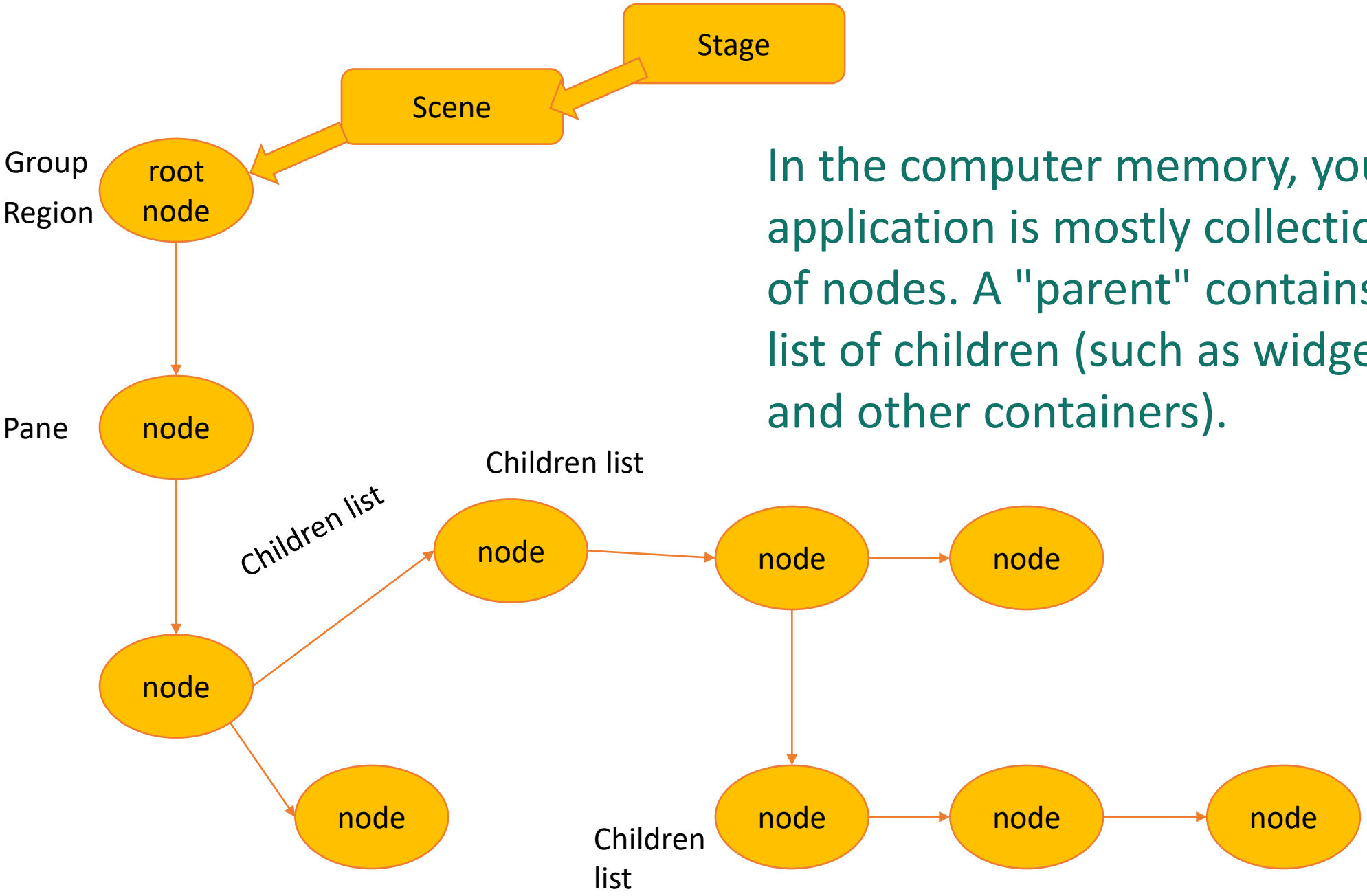
For displaying an image.

UI controls such as `Label`, `TextField`, `Button`, `CheckBox`, `RadioButton`, and `TextArea` are subclasses of Control.

Nodes can be shapes, image views, UI controls, groups, and panes.



Panes and groups are used to hold nodes.



In the computer memory, your application is mostly collections of nodes. A "parent" contains a list of children (such as widgets, and other containers).

Typical Design

```
public static void start(Stage stage) {
        stage.setTitle("Window Title");
        Group root = new Group();
        Scene scene = new Scene(root);
        BorderPane pane = new BorderPane();
        root.getChildren().add(pane);

        // Add containers and widgets to pane

        stage.setScene(scene);
        stage.show();
```

Here is a basic start method for a javaFX program

Panes

ShowFlowPane.java

```
1  import javafx.application.Application;
2  import javafx.geometry.Insets;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Label;
5  import javafx.scene.control.TextField;
6  import javafx.scene.layout.FlowPane;
7  import javafx.stage.Stage;
8
9  public class ShowFlowPane extends Application {
10   @Override // Override the start method in the Application class
11   public void start(Stage primaryStage) {
12     // Create a pane and set its properties
13     FlowPane pane = new FlowPane();
14     pane.setPadding(new Insets(11, 12, 13, 14));
15     pane.setHgap(5);
16     pane.setVgap(5);
17
18     // Place nodes in the pane
19     pane.getChildren().addAll(new Label("First Name:"),
20       new TextField(), new Label("MI:"));
21     TextField tfMi = new TextField();
22     tfMi.setPrefColumnCount(1);
23     pane.getChildren().addAll(tfMi, new Label("Last Name:"),
24       new TextField());
25
26     // Create a scene and place it in the stage
27     Scene scene = new Scene(pane, 200, 250);
28     primaryStage.setTitle("ShowFlowPane"); // Set the stage title
29     primaryStage.setScene(scene); // Place the scene in the stage
30     primaryStage.show(); // Display the stage
31   }
32 }
```

extend Application

create FlowPane

add UI controls to pane

add pane to scene

place scene to stage

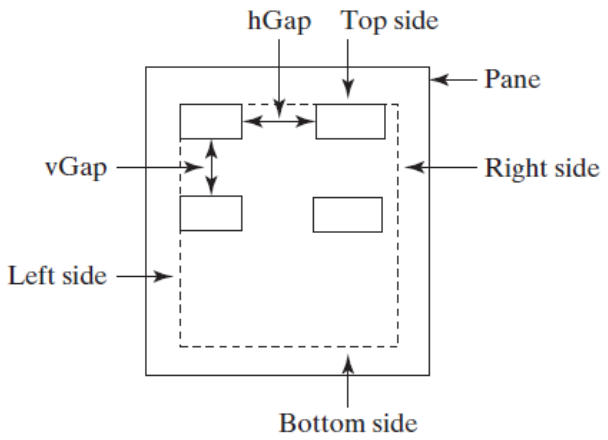
display stage

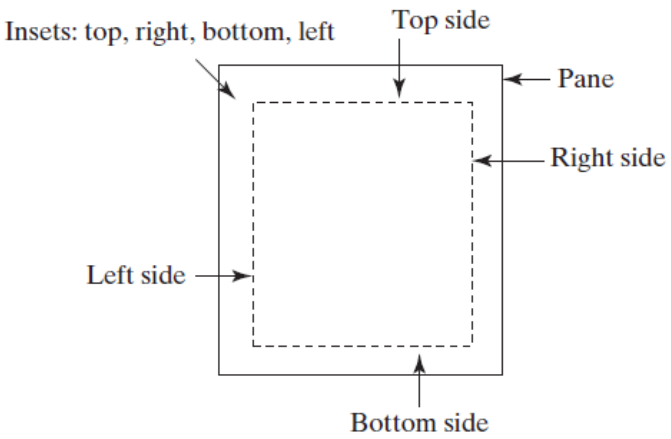
Panes

```
howFlowPane.java

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class ShowFlowPane extends Applicat
  @Override // Override the start method
  public void start(Stage primaryStage) {
    // Create a pane and set its propertie
    FlowPane pane = new FlowPane();
    pane.setPadding(new Insets(11, 12, 13, 14));
    pane.setHgap(5);
    pane.setVgap(5);

    // Place nodes in the pane
    pane.getChildren().addAll(new Label("First Name:"),
      new TextField(), new Label("MI:"));
    TextField tfMi = new TextField();
    tfMi.setPrefColumnCount(1);
    pane.getChildren().addAll(tfMi, new Label("Last Name
      new TextField());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 250);
    primaryStage.setTitle("ShowFlowPane"); // Set the st
    primaryStage.setScene(scene); // Place the scene in
    primaryStage.show(); // Display the stage
  }
}
```

(a) (b)

The nodes fill in the rows in the `FlowPane` one after another.





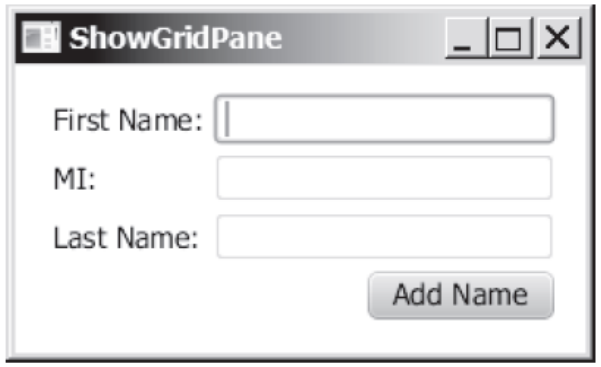
You can specify `hGap` and `vGap` between the nodes in a `FlowLPane`.

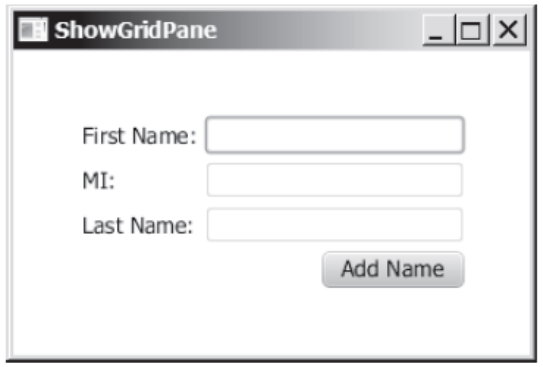
Panes

ShowGridPane.java

```
1  import javafx.application.Application;
2  import javafx.geometry.HPos;
3  import javafx.geometry.Insets;
4  import javafx.geometry.Pos;
5  import javafx.scene.Scene;
6  import javafx.scene.control.Button;
7  import javafx.scene.control.Label;
8  import javafx.scene.control.TextField;
9  import javafx.scene.layout.GridPane;
10 import javafx.stage.Stage;
11
12 public class ShowGridPane extends Application {
13   @Override // Override the start method in the Application class
14   public void start(Stage primaryStage) {
15     // Create a pane and set its properties
16     GridPane pane = new GridPane();
17     pane.setAlignment(Pos.CENTER);
18     pane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
19     pane.setHgap(5.5);
20     pane.setVgap(5.5);
21
22     // Place nodes in the pane
23     pane.add(new Label("First Name:"), 0, 0);
24     pane.add(new TextField(), 1, 0);
25     pane.add(new Label("MI:"), 0, 1);
26     pane.add(new TextField(), 1, 1);
27     pane.add(new Label("Last Name:"), 0, 2);
28     pane.add(new TextField(), 1, 2);
29     Button btAdd = new Button("Add Name");
30     pane.add(btAdd, 1, 3);
31     GridPane.setHalignment(btAdd, HPos.RIGHT);
32
33     // Create a scene and place it in the stage
34     Scene scene = new Scene(pane);
35     primaryStage.setTitle("ShowGridPane"); // Set the stage title
36     primaryStage.setScene(scene); // Place the scene in the stage
37     primaryStage.show(); // Display the stage
38   }
39 }
```

create a grid pane
set properties
add label
add text field
add button
align button right
create a scene
display stage





The `GridPane` places the nodes in a grid with a specified column and row indices.

Panes

ShowBorderPane.java

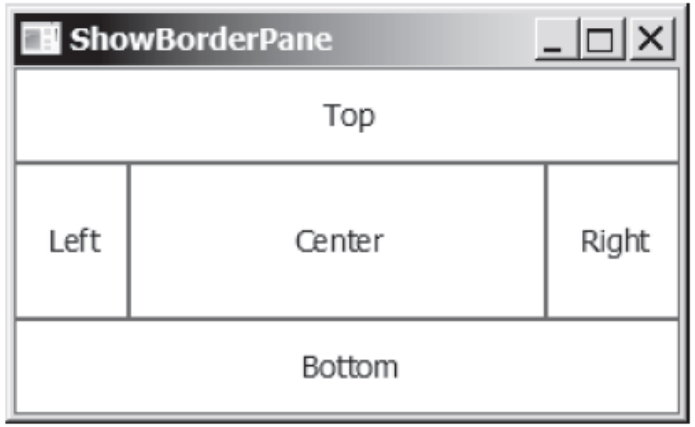
```
1  import javafx.application.Application;
2  import javafx.geometry.Insets;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Label;
5  import javafx.scene.layout.BorderPane;
6  import javafx.scene.layout.StackPane;
7  import javafx.stage.Stage;
8
9  public class ShowBorderPane extends Appli
10   @Override // Override the start method
11   public void start(Stage primaryStage) {
12     // Create a border pane
13     BorderPane pane = new BorderPane();
14
15     // Place nodes in the pane
16     pane.setTop(new CustomPane("Top"));
17     pane.setRight(new CustomPane("Right"));
18     pane.setBottom(new CustomPane("Bottom"));
19     pane.setLeft(new CustomPane("Left"));
20     pane.setCenter(new CustomPane("Center"));
21
22     // Create a scene and place it in the stage
23     Scene scene = new Scene(pane);
24     primaryStage.setTitle("ShowBorderPane"); // Set the stage title
25     primaryStage.setScene(scene); // Place the scene in the stage
26     primaryStage.show(); // Display the stage
27   }
28 }
29
30 // Define a custom pane to hold a label in the center of the pane
31 class CustomPane extends StackPane {
32   public CustomPane(String title) {
33     getChildren().add(new Label(title));
34     setStyle("-fx-border-color: red");
35     setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
36   }
37 }
```

create a border pane

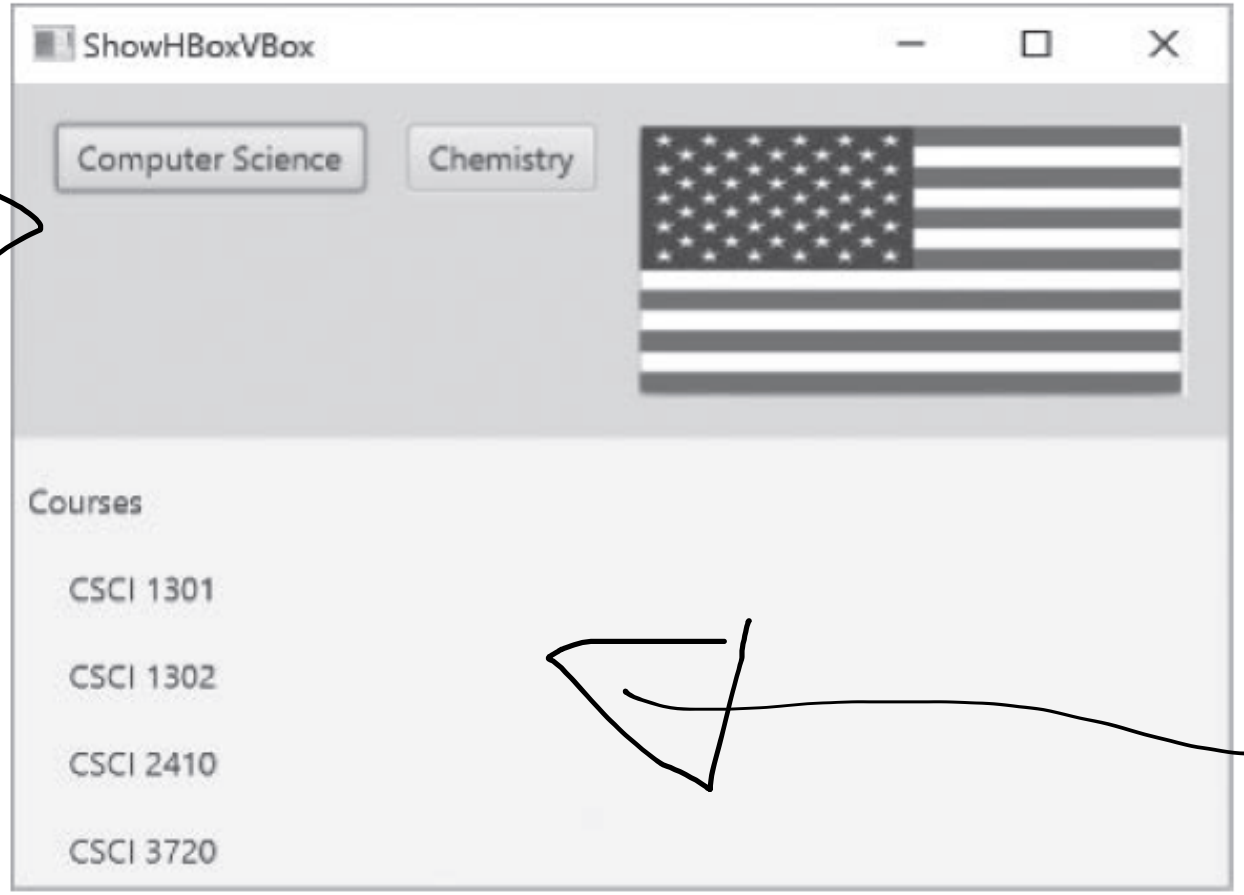
add to top
add to right
add to bottom
add to left
add to center

define a custom pane

add a label to pane
set style
set padding



The BorderPane places the nodes in five regions of the pane.



The **HBox** places the nodes in one row, and the **VBox** places the nodes in one column.

ShowHBoxVBox.java

```
1  import javafx.application.Application;
2  import javafx.geometry.Insets;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Button;
5  import javafx.scene.control.Label;
6  import javafx.scene.layout.BorderPane;
7  import javafx.scene.layout.HBox;
8  import javafx.scene.layout.VBox;
9  import javafx.stage.Stage;
10 import javafx.scene.image.Image;
11 import javafx.scene.image.ImageView;
12
13 public class ShowHBoxVBox extends Application {
14   @Override // Override the start method in the Application class
15   public void start(Stage primaryStage) {
16     // Create a border pane
17     BorderPane pane = new BorderPane();
18
19     // Place nodes in the pane
20     pane.setTop(getHBox());
21     pane.setLeft(getVBox());
22
23     // Create a scene and place it in the stage
24     Scene scene = new Scene(pane);
25     primaryStage.setTitle("ShowHBoxVBox"); // Set the stage title
26     primaryStage.setScene(scene); // Place the scene in the stage
27     primaryStage.show(); // Display the stage
28   }
```

create a border pane

add an HBox to top

add a VBox to left

create a scene

display stage

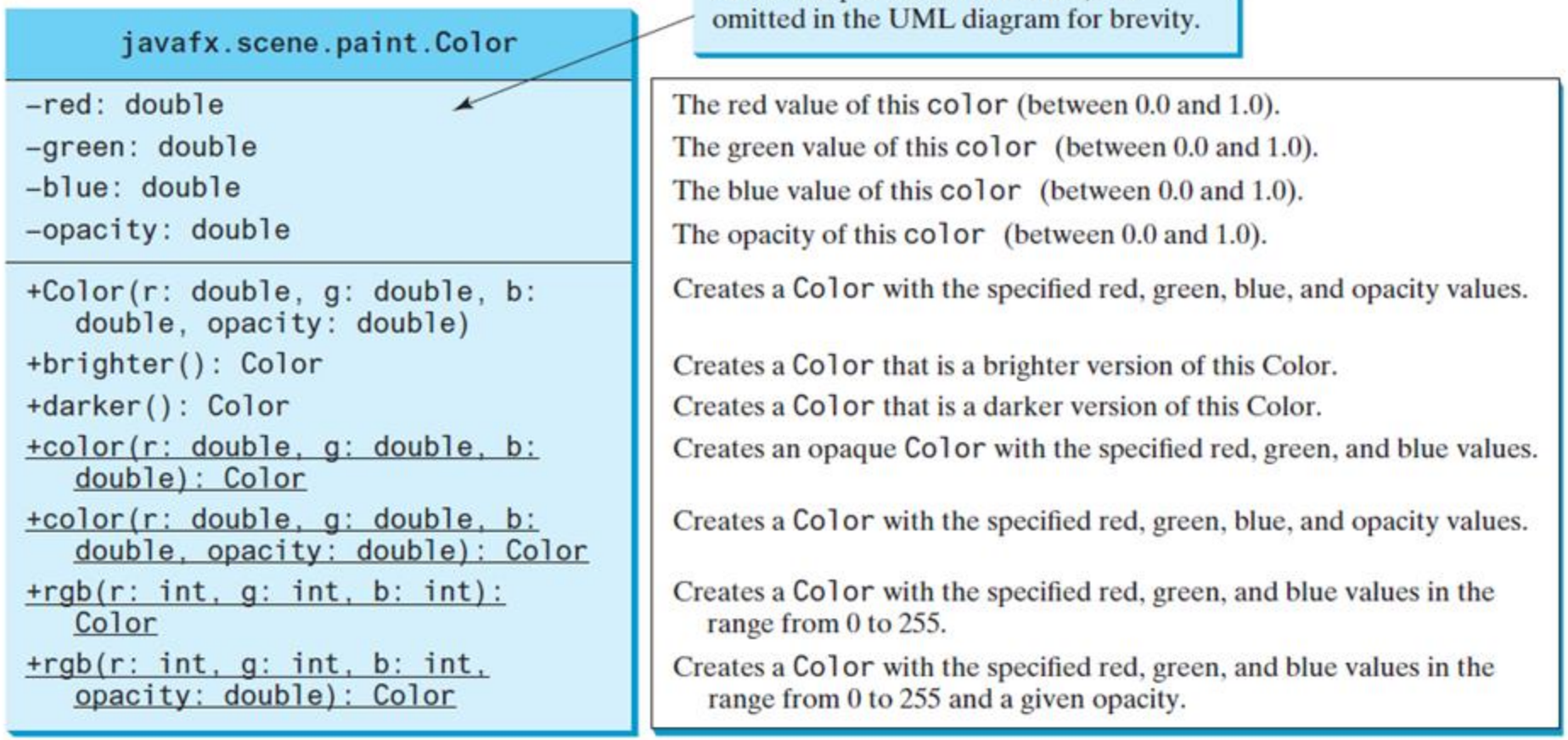
```
                29
getHBox         30    private HBox getHBox() {
                31      HBox hBox = new HBox(15);
                32      hBox.setPadding(new Insets(15, 15, 15, 15));
                33      hBox.setStyle("-fx-background-color: gold");
add buttons to HBox 34  hBox.getChildren().add(new Button("Computer Science"));
                35      hBox.getChildren().add(new Button("Chemistry"));
                36      ImageView imageView = new ImageView(new Image("image/us.gif"));
                37      hBox.getChildren().add(imageView);
return an HBox  38      return hBox;
                39    }
                40
getVBox         41    private VBox getVBox() {
                42      VBox vBox = new VBox(15);
                43      vBox.setPadding(new Insets(15, 5, 5, 5));
add a label     44      vBox.getChildren().add(new Label("Courses"));
                45
                46      Label[] courses = {new Label("CSCI 1301"), new Label("CSCI 1302"),
                47          new Label("CSCI 2410"), new Label("CSCI 3720")};
                48
                49      for (Label course: courses) {
set margin      50        VBox.setMargin(course, new Insets(0, 0, 0, 15));
add a label     51        vBox.getChildren().add(course);
                52      }
                53
return vBox     54      return vBox;
                55    }
                56  }
```

The `Color` Class

The `Color` class can be used to create colors.

JavaFX defines the abstract `Paint` class for painting a node. The `javafx.scene.paint.Color` is a concrete subclass of `Paint`.

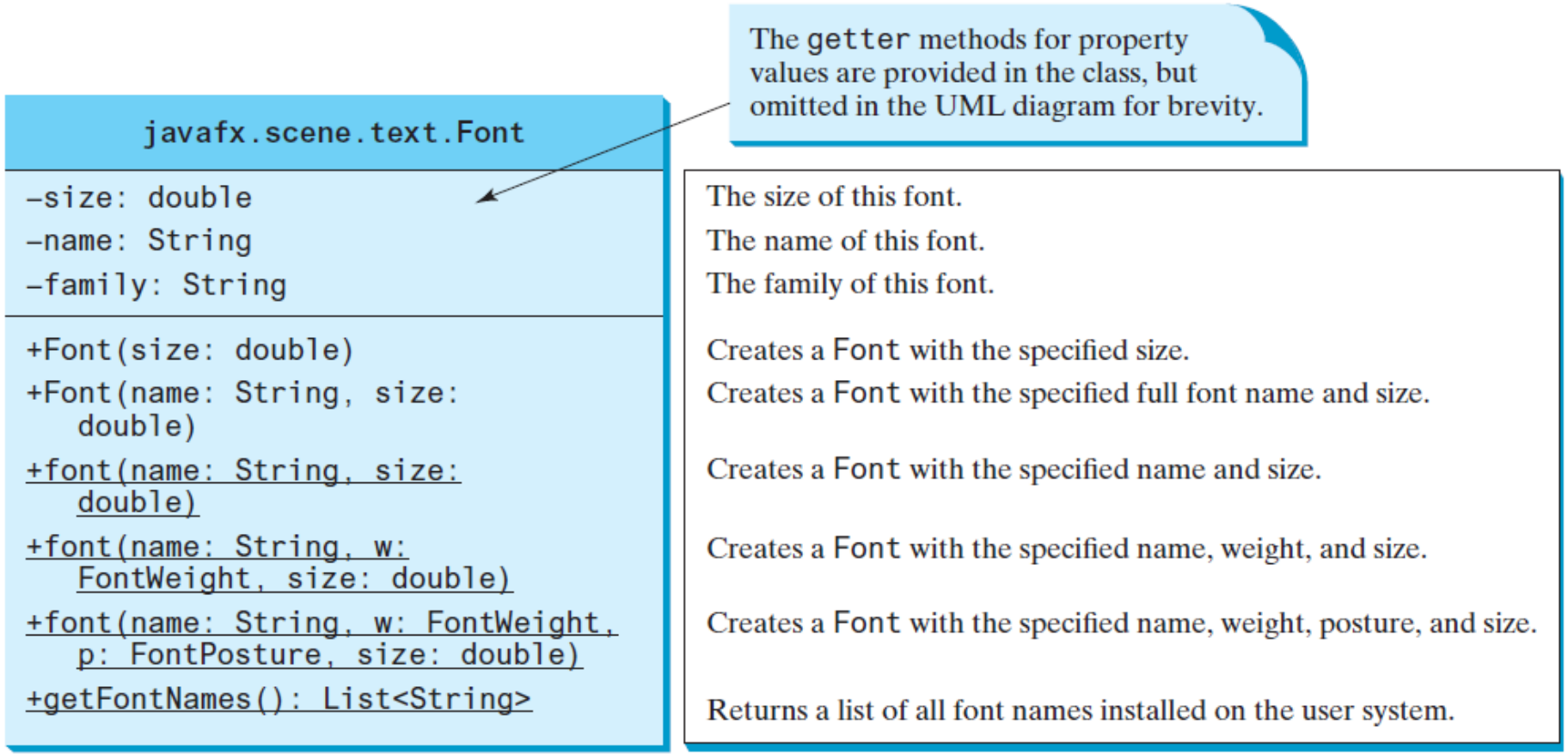
The `getter` methods for property values are provided in the class, but omitted in the UML diagram for brevity.



The Font Class

A `Font` *describes font name, weight, and size.*

```
Font font1 = new Font("SansSerif", 16);
Font font2 = Font.font("Times New Roman", FontWeight.BOLD,
  FontPosture.ITALIC, 12);
```

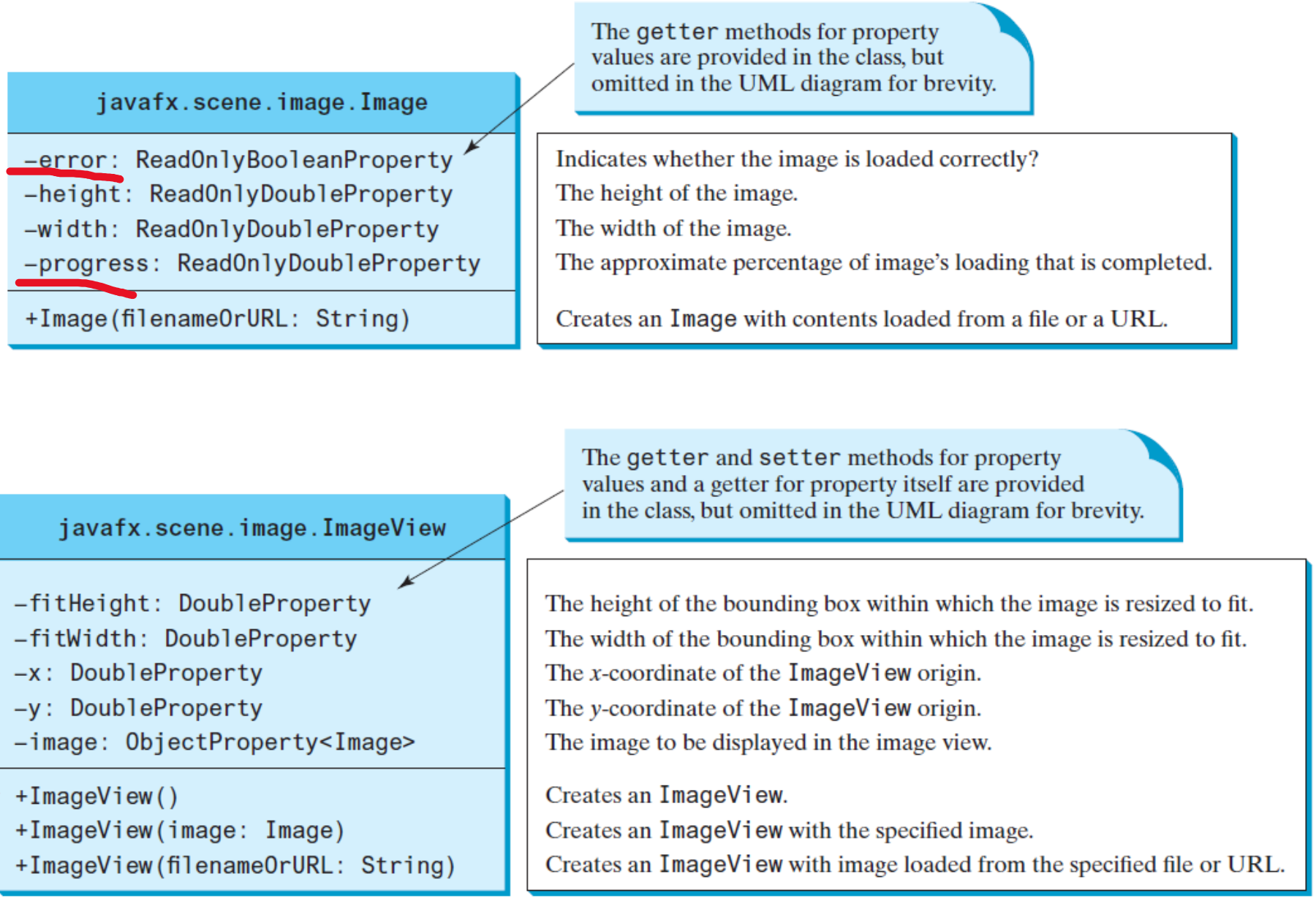


The `Image` and `ImageView` Classes

The `Image` *class represents a graphical image, and the* `ImageView` *class can be used to display an image.*

```
Image image = new Image("image/us.gif");
ImageView imageView = new ImageView(image);
```

```
ImageView imageView = new ImageView("image/us.gif");
```

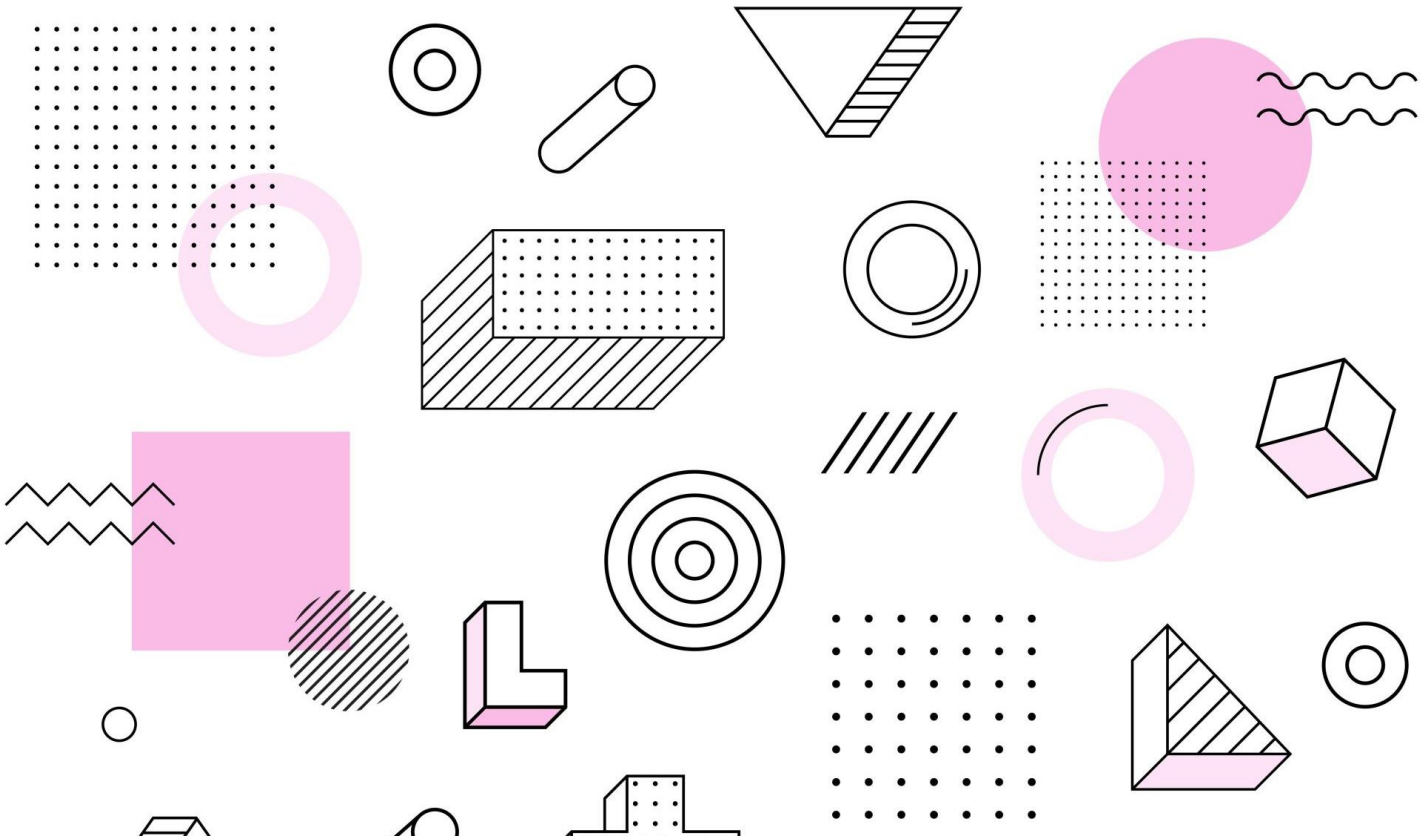

javafx.scene.image.Image
-error: ReadOnlyBooleanProperty
-height: ReadOnlyDoubleProperty
-width: ReadOnlyDoubleProperty
-progress: ReadOnlyDoubleProperty
+Image(filenameOrURL: String)
The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.
Indicates whether the image is loaded correctly?
The height of the image.
The width of the image.
The approximate percentage of image's loading that is completed.
Creates an Image with contents loaded from a file or a URL.
javafx.scene.image.ImageView
-fitHeight: DoubleProperty
-fitWidth: DoubleProperty
-x: DoubleProperty
-y: DoubleProperty
-image: ObjectProperty<Image>
+ImageView()
+ImageView(image: Image)
+ImageView(filenameOrURL: String)
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
The height of the bounding box within which the image is resized to fit.
The width of the bounding box within which the image is resized to fit.
The x-coordinate of the ImageView origin.
The y-coordinate of the ImageView origin.
The image to be displayed in the image view.
Creates an ImageView.
Creates an ImageView with the specified image.
Creates an ImageView with image loaded from the specified file or URL.

ShowImage.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.HBox;
4  import javafx.scene.layout.Pane;
5  import javafx.geometry.Insets;
6  import javafx.stage.Stage;
7  import javafx.scene.image.Image;
8  import javafx.scene.image.ImageView;
9
10 public class ShowImage extends Application {
11   @Override // Override the start method in the Application class
12   public void start(Stage primaryStage) {
13     // Create a pane to hold the image views
14     Pane pane = new HBox(10);                          create an HBox
15     pane.setPadding(new Insets(5, 5, 5, 5));
16     Image image = new Image("image/us.gif");           create an image
17     pane.getChildren().add(new ImageView(image));      add an image view to pane
18
19     ImageView imageView2 = new ImageView(image);       create an image view
20     imageView2.setFitHeight(100);                      set image view properties
21     imageView2.setFitWidth(100);
22     pane.getChildren().add(imageView2);                add an image to pane
23
24     ImageView imageView3 = new ImageView(image);       create an image view
25     imageView3.setRotate(90);                          rotate an image view
26     pane.getChildren().add(imageView3);                add an image to pane
27
28     // Create a scene and place it in the stage
29     Scene scene = new Scene(pane);
30     primaryStage.setTitle("ShowImage"); // Set the stage title
31     primaryStage.setScene(scene); // Place the scene in the stage
32     primaryStage.show(); // Display the stage
33   }
34 }
```

Next

- More widgets and other components for controlling interactions…

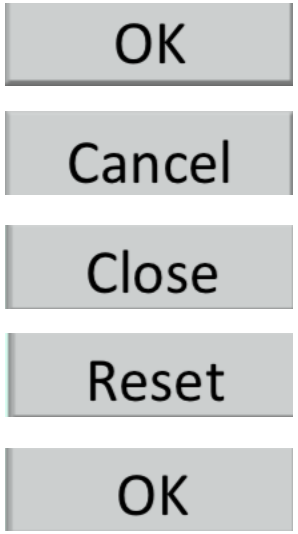


Graphical User Interface V

Various Widgets

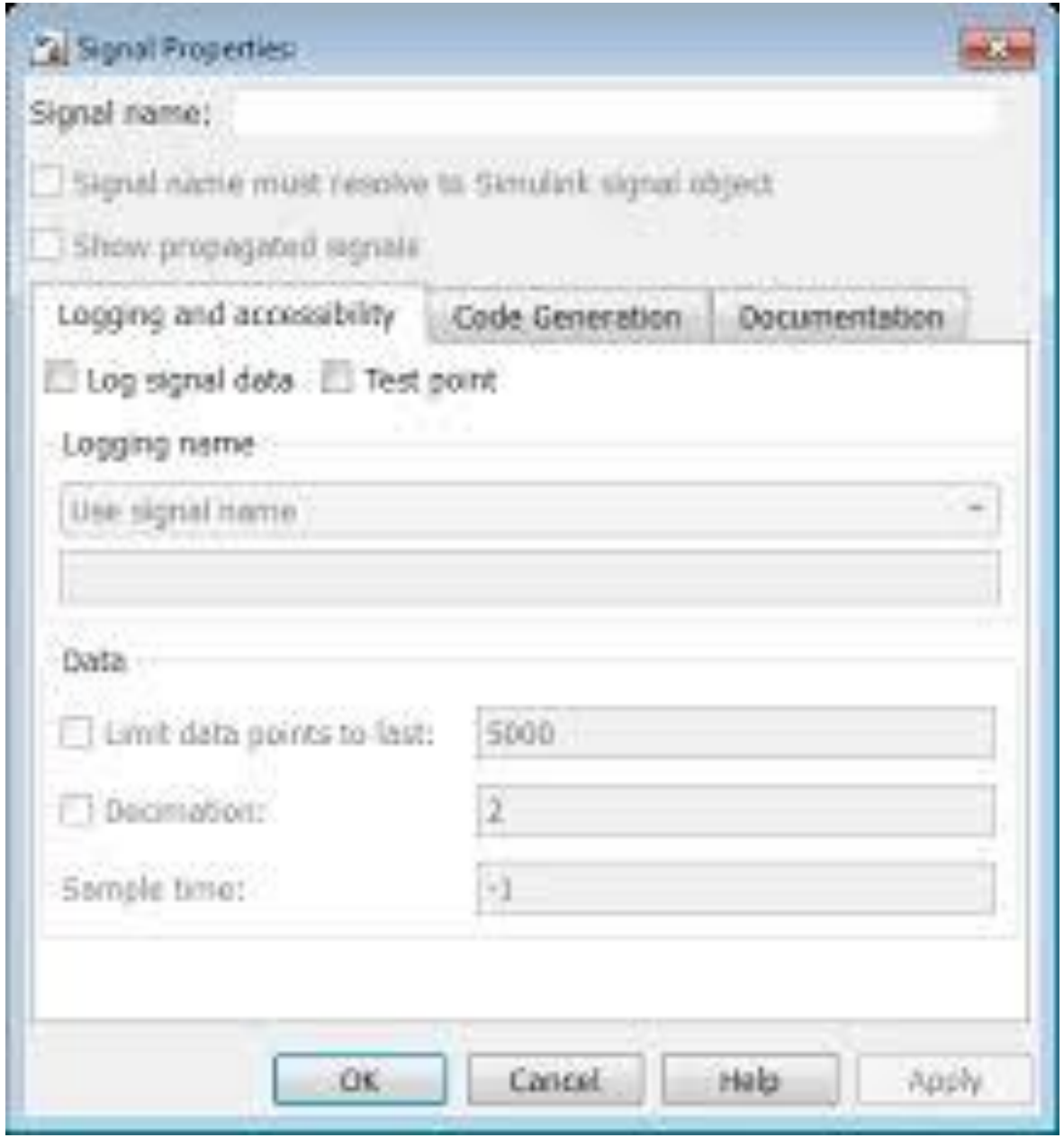
Widgets: Buttons

Common buttons and expected behavior…



Standards are important in making a GUI usable, and as a result your application, consistence also creates a professional appearance.

- Changes applied, close window
- No changes, close window
- Can’t cancel, close window
- Set default, keep window open
- Sometimes changes applied, keep window open


Signal Properties
Signal name:
Signal name must resolve to Simulink signal object
Show propagated signals
Logging and accessibility
Code Generation
Documentation
Log signal data
Test point
Logging name
Use signal name
Data
Limit data points to last:
5000
Decimation:
2
Sample time:
-1
OK
Cancel
Help
Apply

Widgets: Buttons

Keep all buttons the same size

... or have a "short button" and a "long" button size

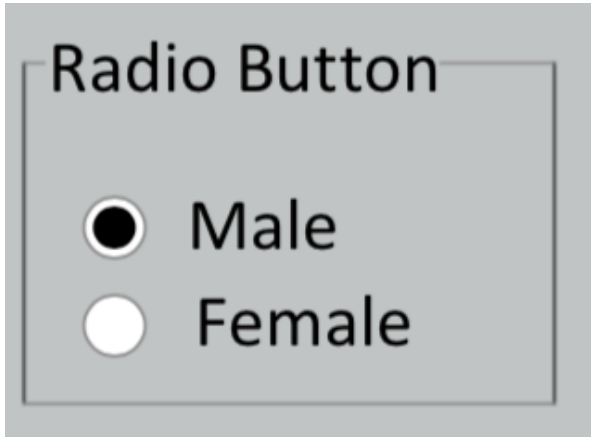
Group buttons

Isolate buttons from the rest (space)

Widgets: Radio Buttons

For several exclusive choices

Usually in a group

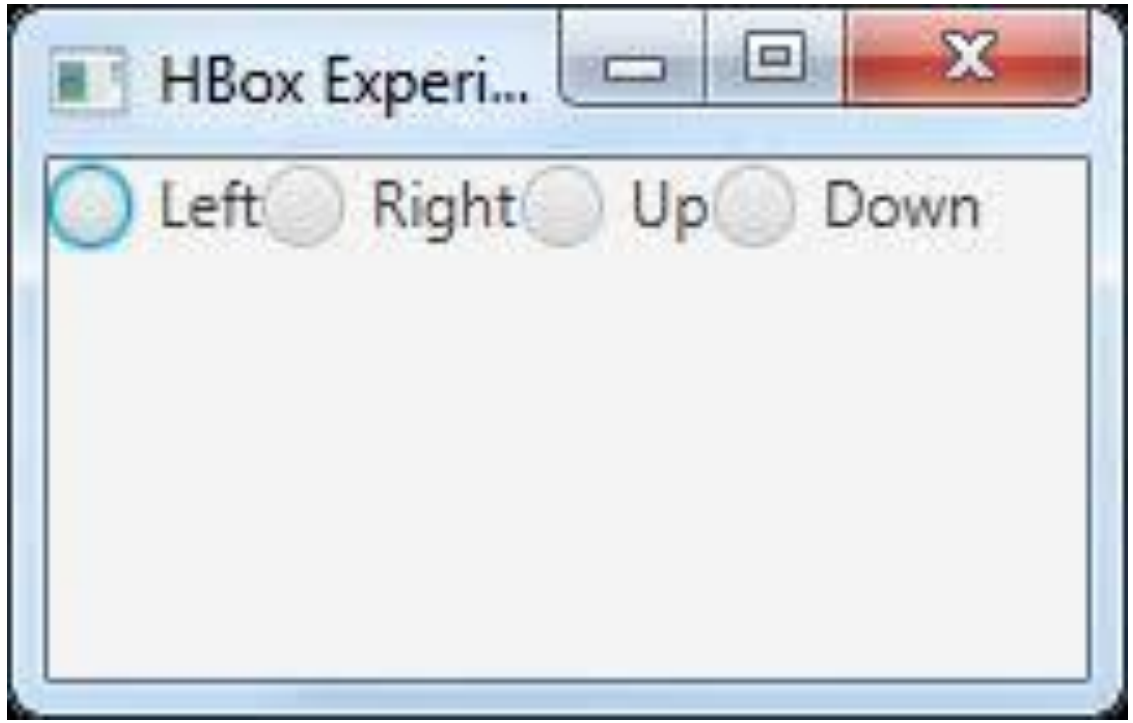


In a quiz a radio button will tell you only one answer is correct….

Toggle group object

Toggle groups can be used to make radio buttons represent a set of on off switches in which only one can be on

javafx.scene.control.ToggleGroup

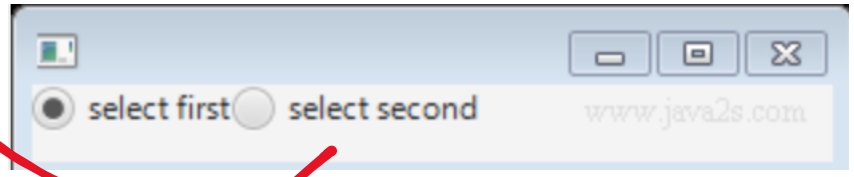


Toggle group object

Set of on off switches in which one can be on.

javafx.scene.control.ToggleGroup

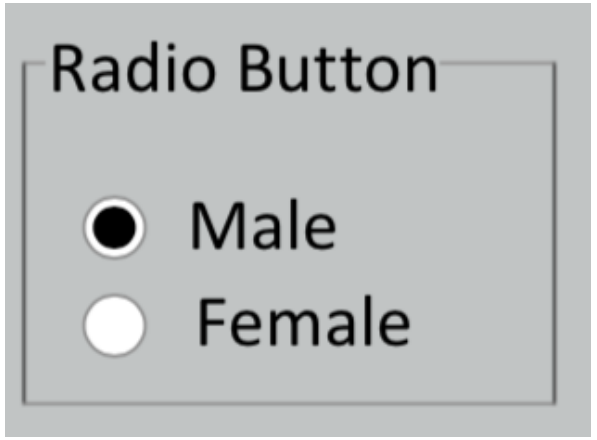
```
ToggleGroup radioGroup = new ToggleGroup();
radioButton1.setToggleGroup(radioGroup);
radioButton2.setToggleGroup(radioGroup);
radioButton3.setToggleGroup(radioGroup);
```

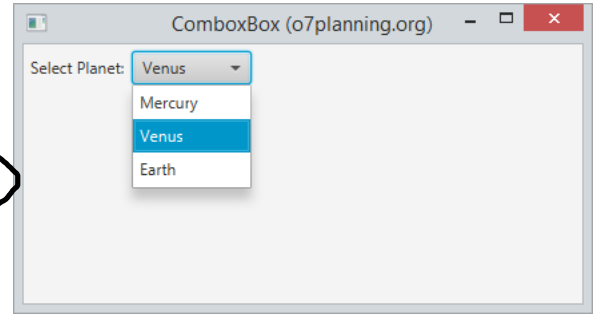



```
public void start(Stage stage) {
  HBox root = new HBox();
  Scene scene = new Scene(root, 300, 150);
  stage.setScene(scene);
  stage.setTitle("");

  ToggleGroup group = new ToggleGroup();
  RadioButton button1 = new RadioButton("select first");
  button1.setToggleGroup(group);
  button1.setSelected(true);
  RadioButton button2 = new RadioButton("select second");
  button2.setToggleGroup(group);

  root.getChildren().add(button1);
  root.getChildren().add(button2);

  scene.setRoot(root);
  stage.show();
}
```

Radio Button Widgets

- One of several exclusive choices
- Usually in a group
- Use vertically
- Six options or less
- If more than six options use a ListBox
- Avoid Yes/No or On/Off

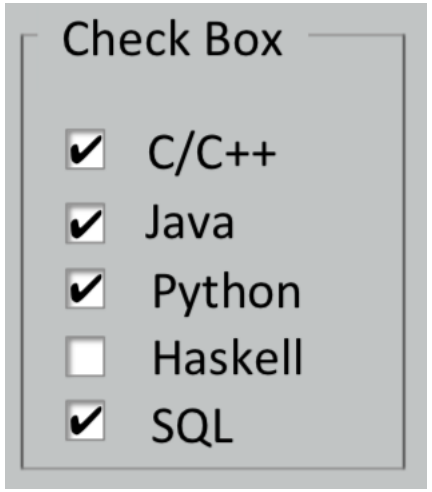




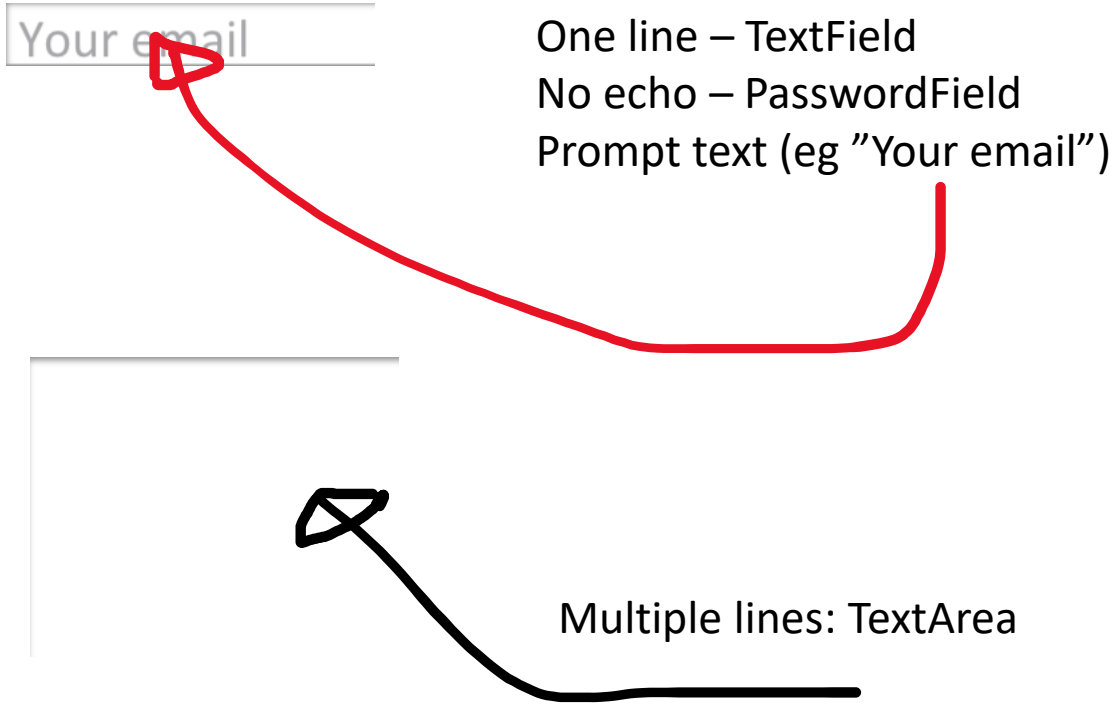
Widgets: CheckBox

- More than several options allowed
- Toggling (Yes/No or On/Off)
- Use vertically
- Ten options or less
- Button for ”select all”

- Alternative is a multiple – select ListBox

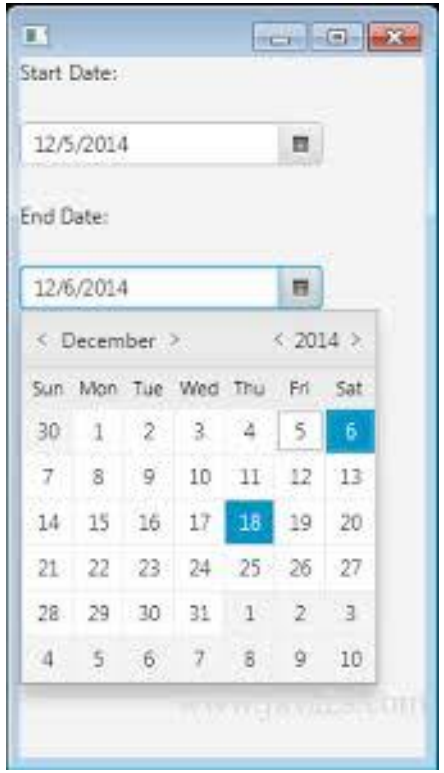


Widgets for getting data from the user

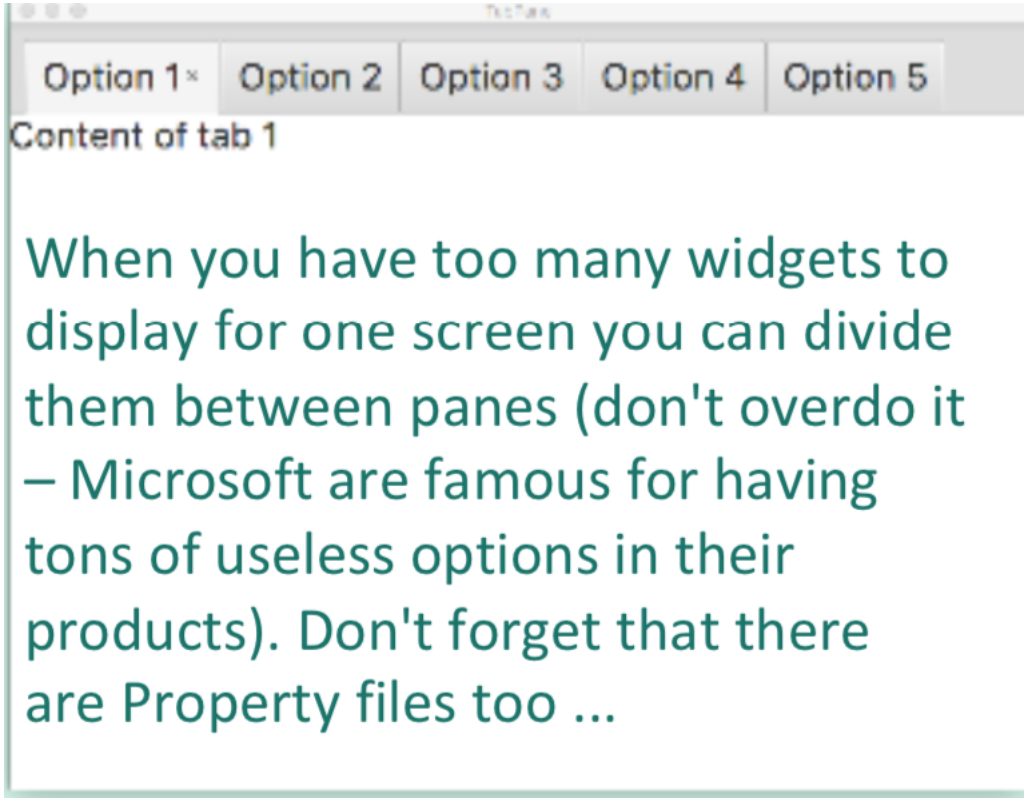


Special Widgets for Special Purposes

- ColorPicker
- DatePicker
- There are others too…



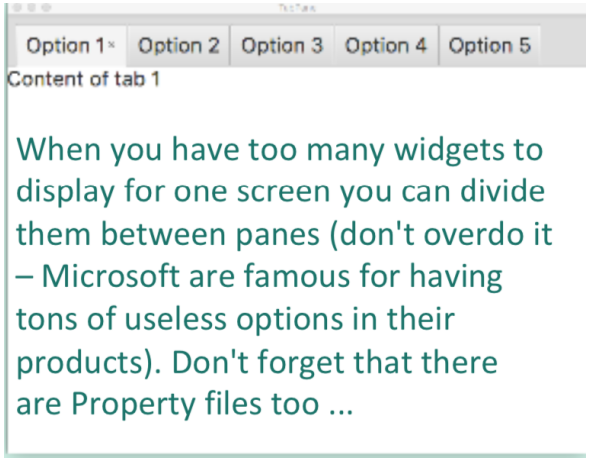
Widget TabPane



- TabPane: useful to avoid clutter

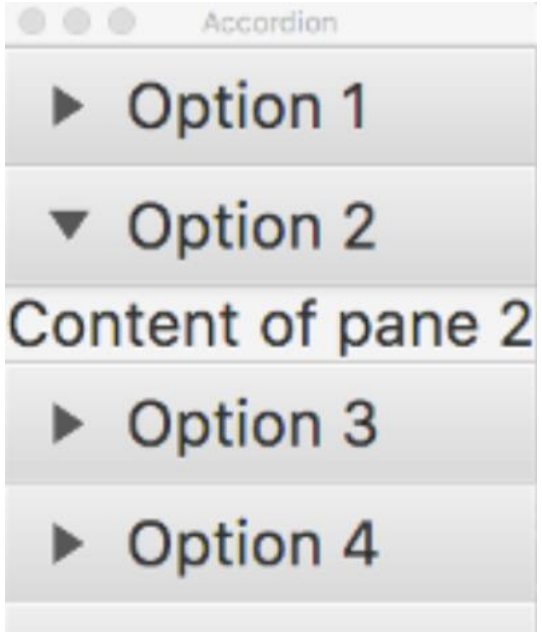
```
...
// Create a TabPane
TabPane pane = new TabPane();
pane.setPrefWidth(800);
pane.setPrefHeight(600);
root.getChildren().add(pane);
Tab tab;


// Create five tabs
for (int i = 1; i <= 5; i++) {
            tab = new Tab();
            tab.setText("Option " + i);
            tab.setContent(new Label("Content of tab " + i));
            pane.getTabs().add(tab);
            }
```

Container of tabs



Nodes can have any container or other widget

Widget Accordion and Titled Panes



- Titled panes are added to an accordion

Too Many Widgets? Can Use Accordion and Titled Panes

```
// Create an Accordion
Accordion accordion = new Accordion();
root.getChildren().add(accordion);
TitledPane pane;


// Create five titled panes
for (int i = 1; i <= 5; i++) {
        pane = new TitledPane();
        pane.setText("Option " + i);
        pane.setContent(new Label("Content of pane " + i));
        accordion.getPanes().add(pane);
        }
```

Padding and Spacing

Padding Distance from the edge

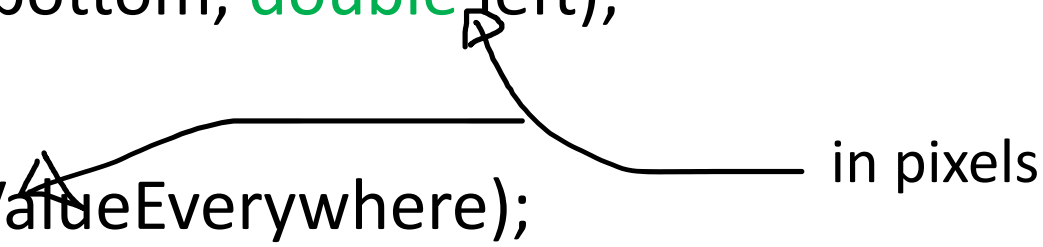
Spacing Distance between widgets

To make everything more readable, there should be “white” space. Two options, padding and spacing (which can change when you resize windows)

Padding and Spacing

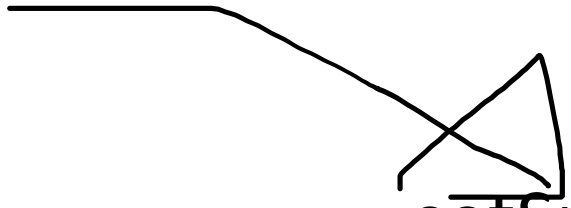
.setPadding(Insets paddingValue)

```
import javafx.geometry.Insets;
Insets(double top, double right,
                double bottom, double left);

Insets(double sameValueEverywhere);
```



Padding and Spacing

- Same distance between all the widgets in a container



.setSpacing(double spacingValue)

- Used to compute initial size
- When the window is first displayed, it may have a size you set, or the size may be computed.
- Of course, a lot of things will change in spacing if you broaden the window for instance.

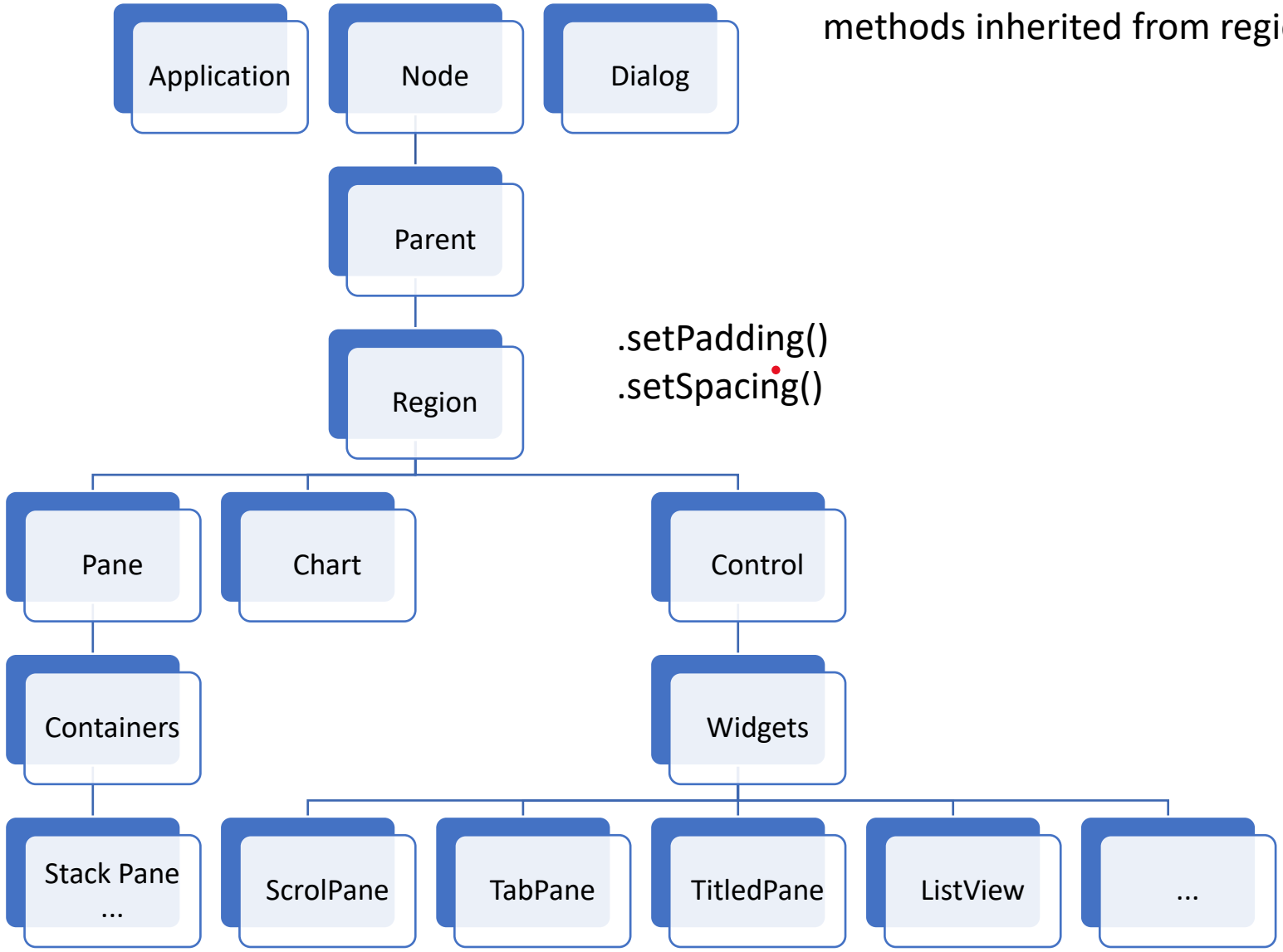
Padding and Spacing

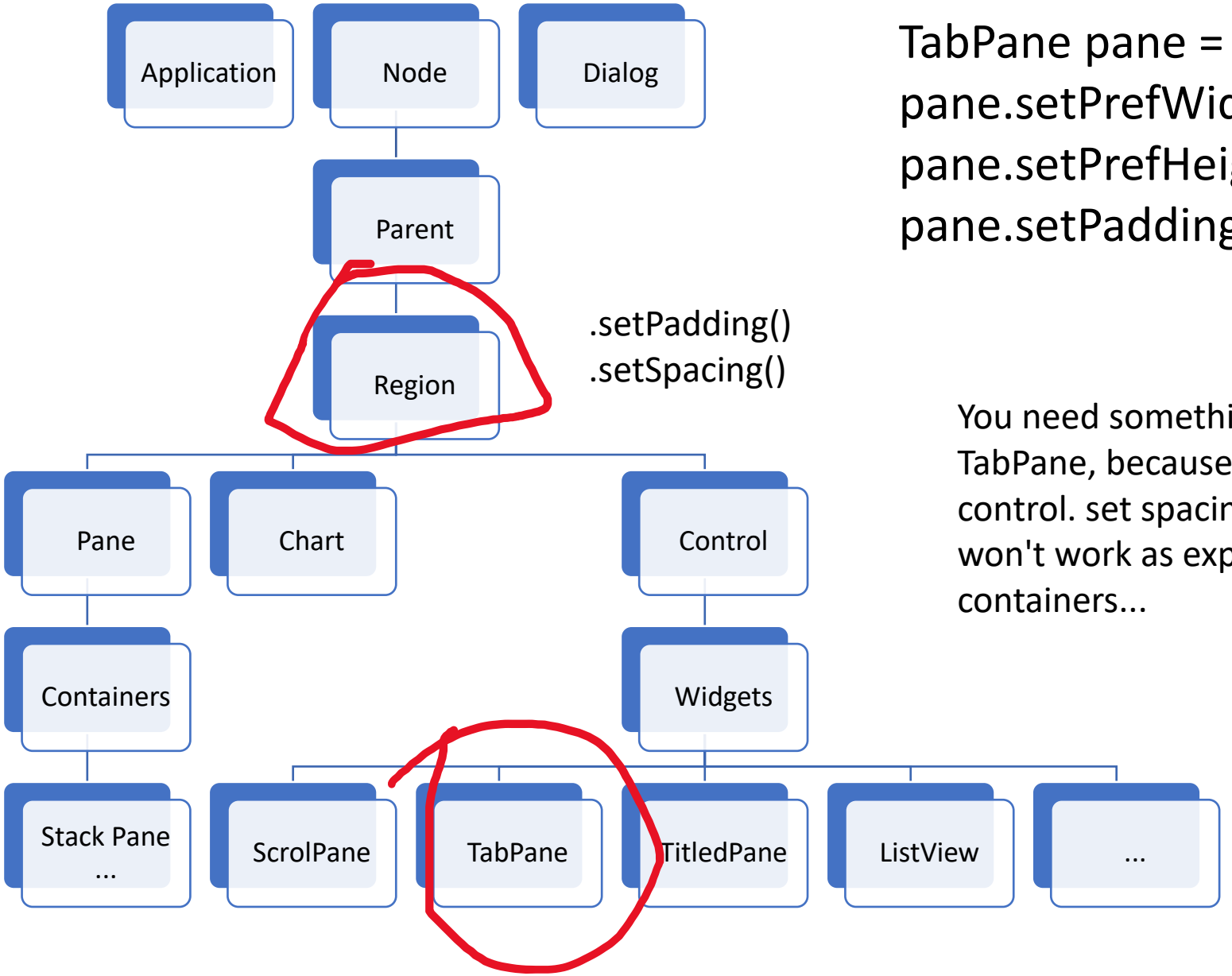
- Some containers (BorderPane, GridPane, HBox, VBox, StackPane, TilePane) implement a static method:

.setMargin(Node child, Insets marginValue)

(allows for setting spacing at the level of the individual widgets)

Be careful: padding and spacing are set with methods inherited from region



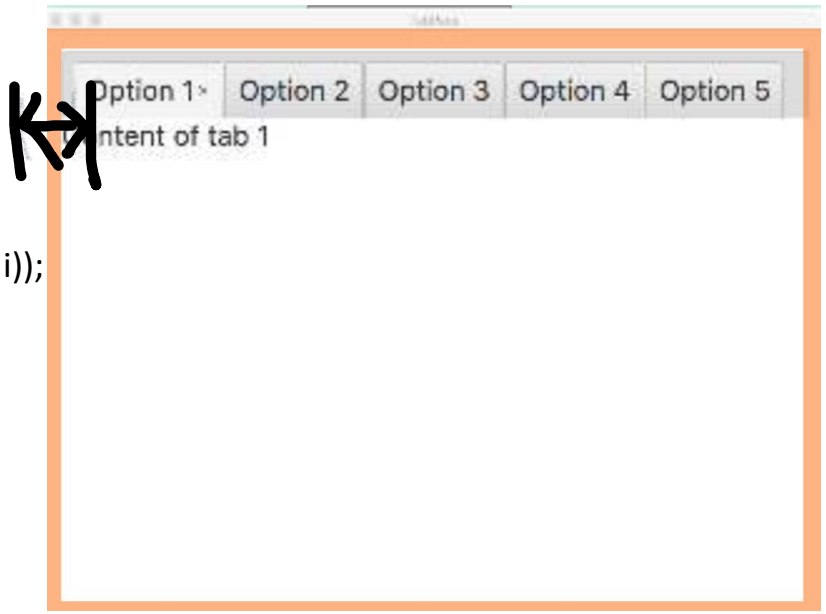



```
TabPane pane = new TabPane();
pane.setPrefWidth(800);
pane.setPrefHeight(600);
pane.setPadding(new Insets(20));
```

You need something special with a TabPane, because it's a composite control. set spacing or padding won't work as expected for containers...

Instead you should add a container (region) to the tab, eg Vbox:

```
Tab tab;
VBox tabBox;
// Create five tabs
for (int i = 1; i <= 5; i++) {
        tab = new Tab(); tab.setText("Option " + i);
        tabBox = new VBox();
        tabBox.setPadding(new Insets(20));
        tabBox.getChildren().add(new Label("Content of tab " + i));
        tab.setContent(tabBox);
        pane.getTabs().add(tab);
        }
```

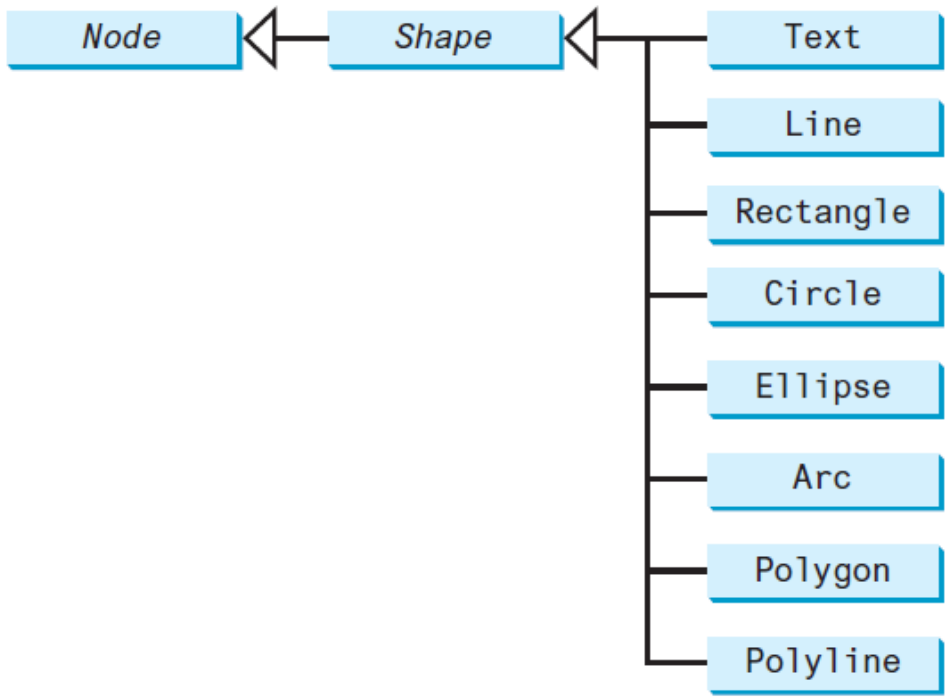


Shapes

Shapes

JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.

The `Shape` class is the abstract base class that defines the common properties for all shapes. Among them are the `fill`, `stroke`, and `strokeWidth` properties. The `fill` property specifies a color that fills the interior of a shape. The `stroke` property specifies a color that is used to draw the outline of a shape. The `strokeWidth` property specifies the width of the outline of a shape. This section introduces the classes `Text`, `Line`, `Rectangle`, `Circle`, `Ellipse`, `Arc`, `Polygon`, and `Polyline` for drawing texts and simple shapes. All these are subclasses of `Shape`, as shown in Figure 14.25.



A shape is a node. The `Shape` class is the root of all shape classes.

The `getter` and `setter` methods for property value and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.text.Text

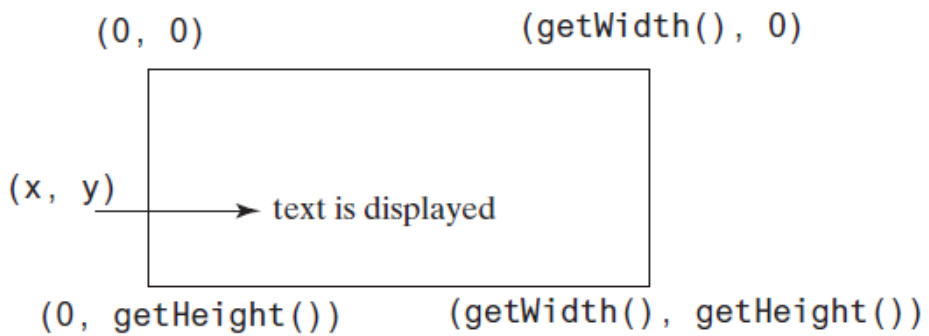
```
-text: StringProperty
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
```

```
+Text()
+Text(text: String)
+Text(x: double, y: double,
   text: String)
```

Defines the text to be displayed.
Defines the *x*-coordinate of text (default 0).
Defines the *y*-coordinate of text (default 0).
Defines if each line has an underline below it (default false).
Defines if each line has a line through it (default false).
Defines the font for the text.

Creates an empty `Text`.
Creates a `Text` with the specified text.
Creates a `Text` with the specified *x*-, *y*-coordinates and text.

`Text` defines a node for displaying a text.



(a) `Text`(*x*, *y*, text)

(b) Three `Text` objects are displayed

A Text object is created to display a text.

The `getter` and `setter` methods for property value and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

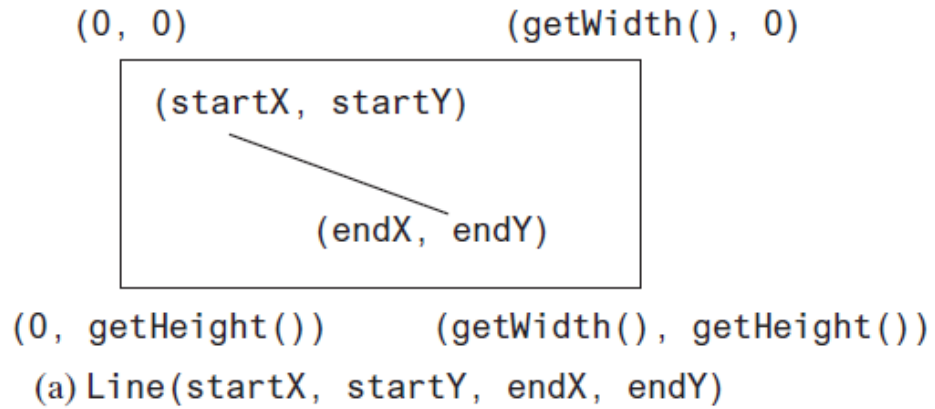
javafx.scene.shape.Line

```
-startX: DoubleProperty
-startY: DoubleProperty
-endX: DoubleProperty
-endY: DoubleProperty

+Line()
+Line(startX: double, startY:
   double, endX: double, endY:
   double)
```

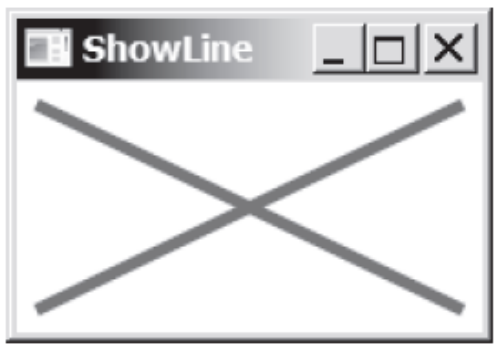
The *x*-coordinate of the start point.
The *y*-coordinate of the start point.
The *x*-coordinate of the end point.
The *y*-coordinate of the end point.

Creates an empty `Line`.
Creates a `Line` with the specified starting and ending points.

The `Line` class defines a line.



(a) `Line(startX, startY, endX, endY)`



(b) Two lines are displayed across the pane.

A `Line` object is created to display a line.

The getter and setter me thods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.Rectangle

```
-x: DoubleProperty
-y: DoubleProperty
-width: DoubleProperty
-height: DoubleProperty
-arcWidth: DoubleProperty

-arcHeight: DoubleProperty
```

```
+Rectangle()
+Rectangle(x: double, y:
   double, width: double,
   height: double)
```

The *x*-coordinate of the upper-left corner of the rectangle (default 0).

The *y*-coordinate of the upper-left corner of the rectangle (default 0).

The `width` of the rectangle (default: 0).

The `height` of the rectangle (default: 0).

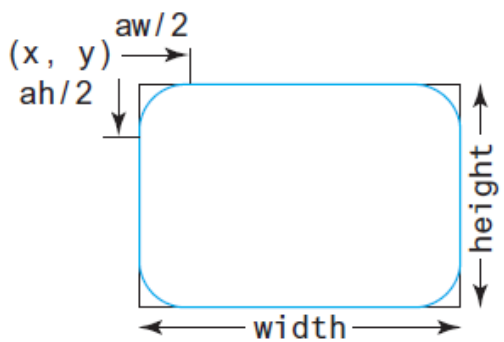
The `arcWidth` of the rectangle (default: 0). arcWidth is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

The `arcWidth` of the rectangle (default: 0). arcHeight is the vertical diameter of the arcs at the corner (see Figure 14.31a).

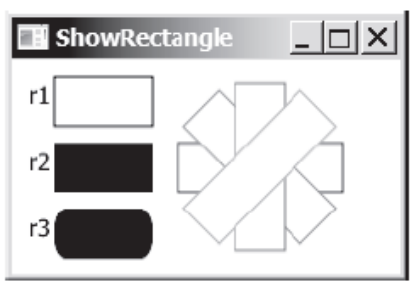
Creates an empty `Rectangle`.

Creates a `Rectangle` with the specified upper-left corner point, width, and height.

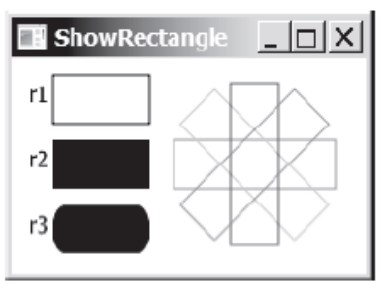
`Rectangle` defines a rectangle.



(a) `Rectangle(x, y, w, h)`

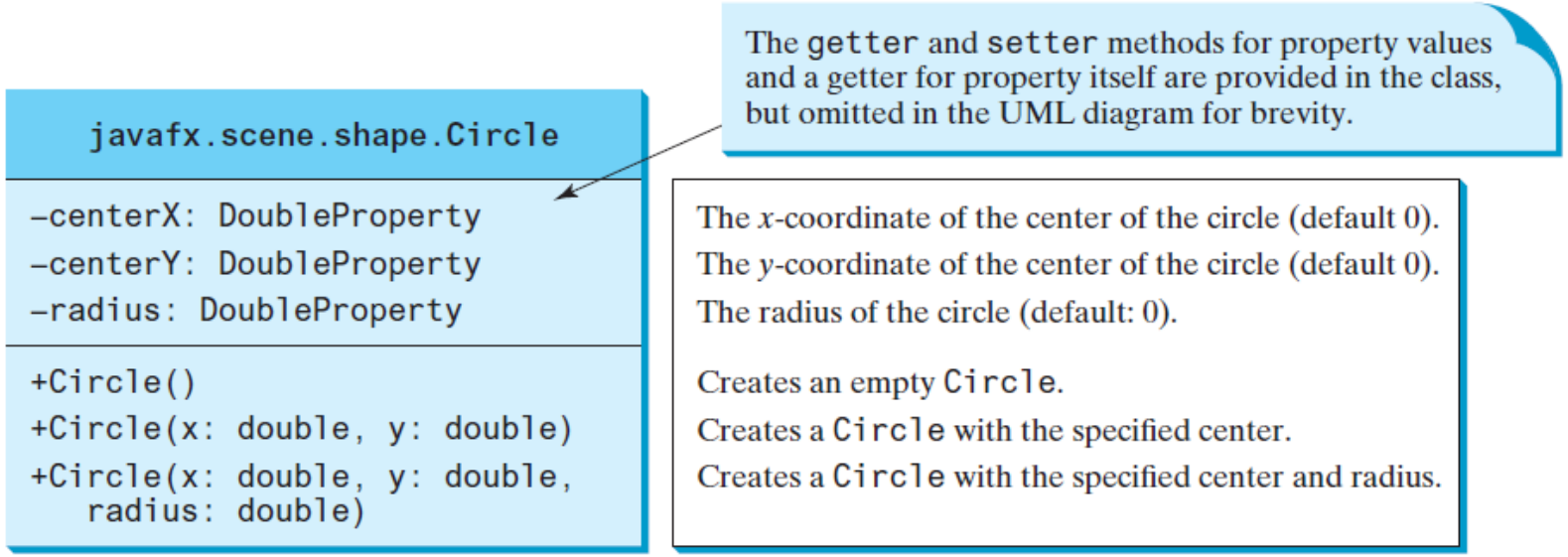


(b) Multiple rectangles are displayed.

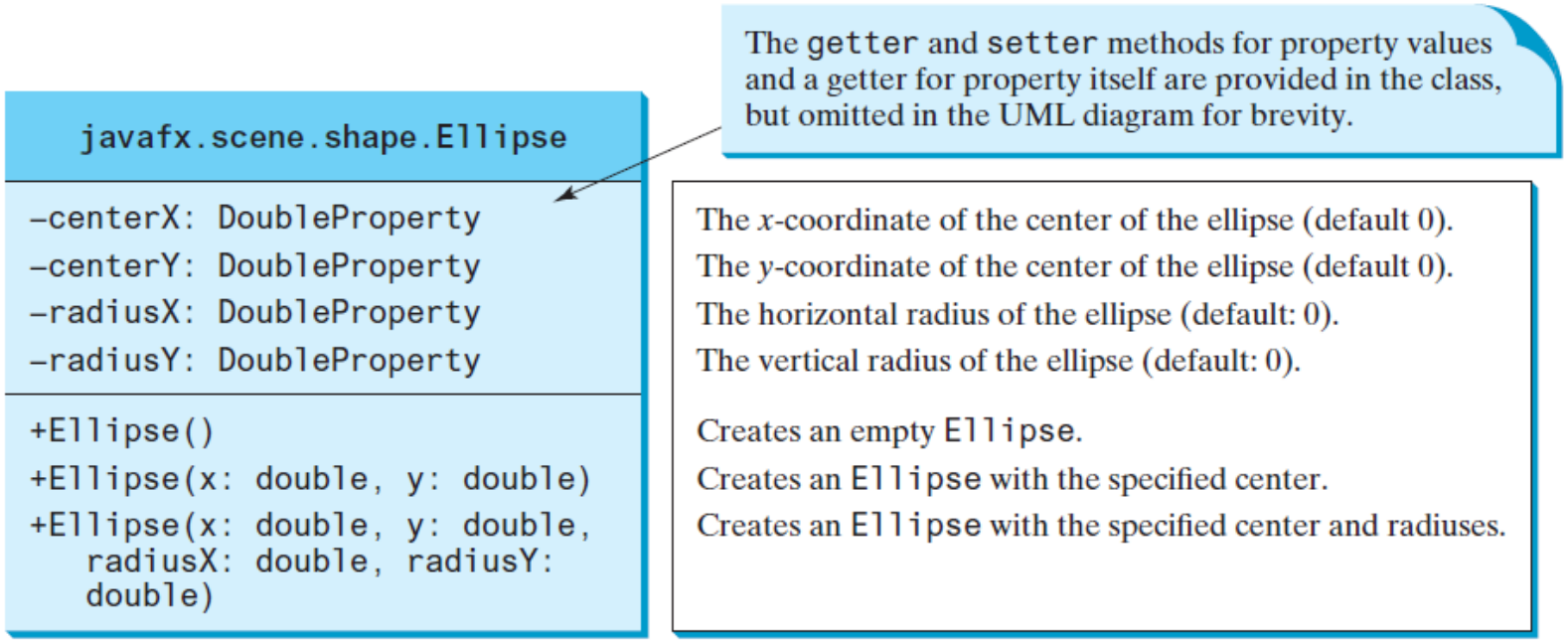


(c) Transparent rectangles are displayed.

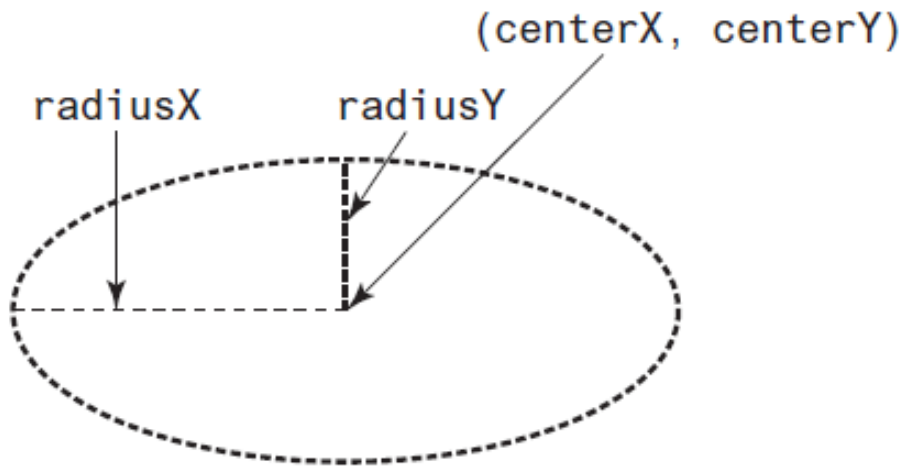
A `Rectangle` object is created to display a rectangle.



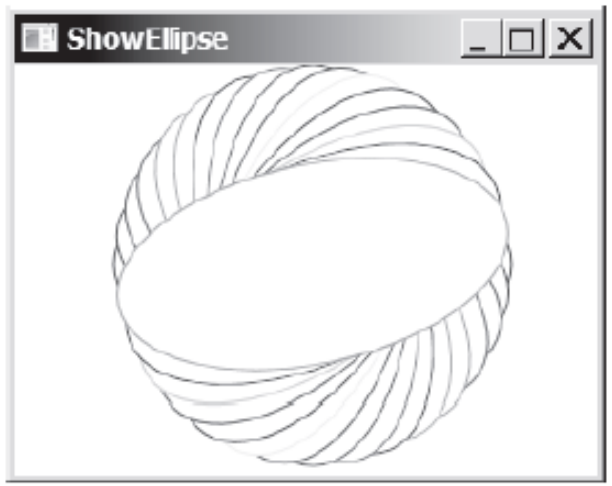
The `Circle` class defines circles.



The `Ellipse` class defines ellipses.



(a) `Ellipse(centerX, centerY, radiusX, radiusY)`



(b) Multiple ellipses are displayed.

An `Ellipse` object is created to display an ellipse.

The `getter` and `setter` methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.Arc

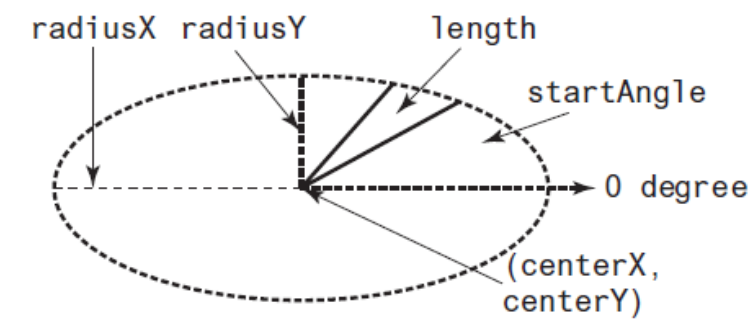
```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
```

```
+Arc()
+Arc(x: double, y: double,
   radiusX: double, radiusY:
   double, startAngle: double,
   length: double)
```

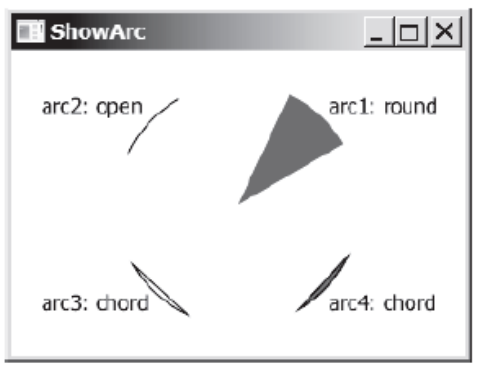
The *x*-coordinate of the center of the ellipse (default 0).
The *y*-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).
The start angle of the arc in degrees.
The angular extent of the arc in degrees.
The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

Creates an empty `Arc`.
Creates an `Arc` with the specified arguments.

The `Arc` class defines an arc.

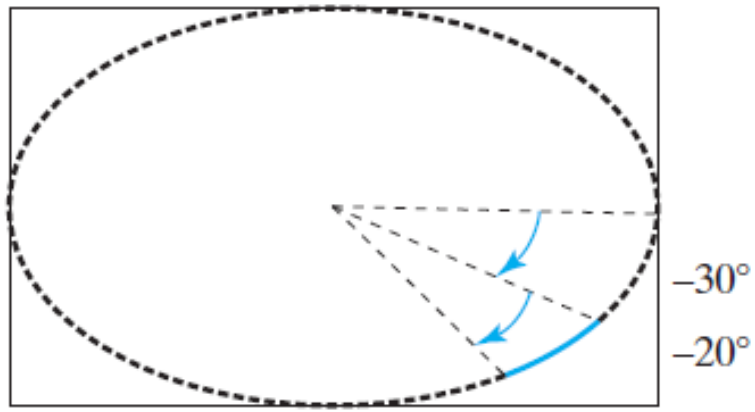


(a) `Arc(centerX, centerY, radiusX, radiusY, startAngle, length)`

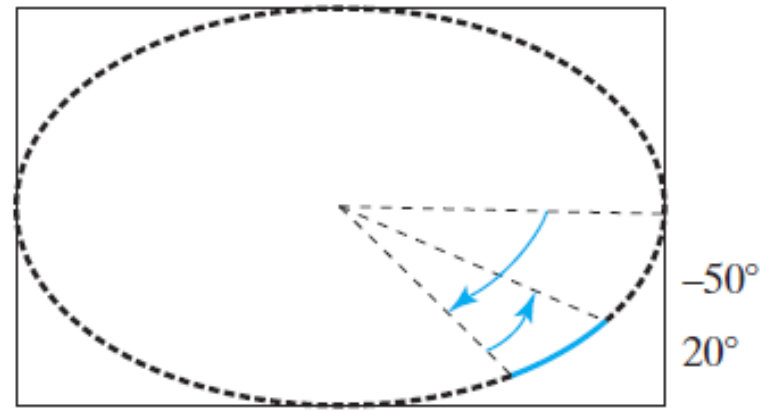


(b) Multiple ellipses are displayed.

An `Arc` object is created to display an arc.

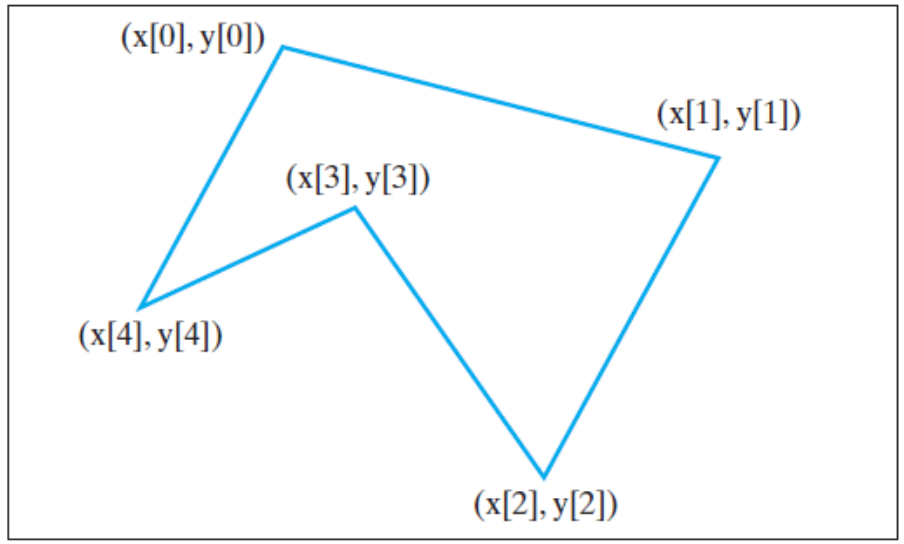


(a) Negative starting angle –30° and negative spanning angle –20°

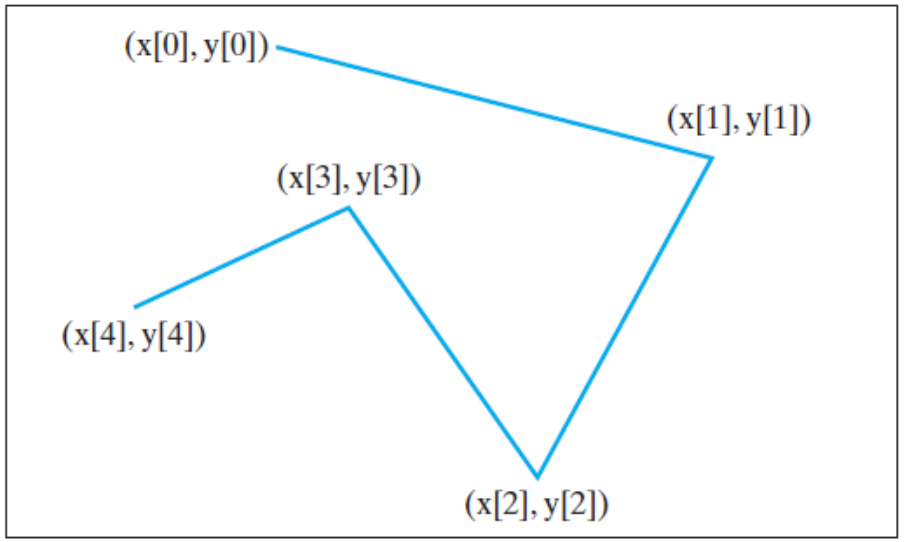


(b) Negative starting angle –50° and positive spanning angle 20°

Angles may be negative.



(a) Polygon

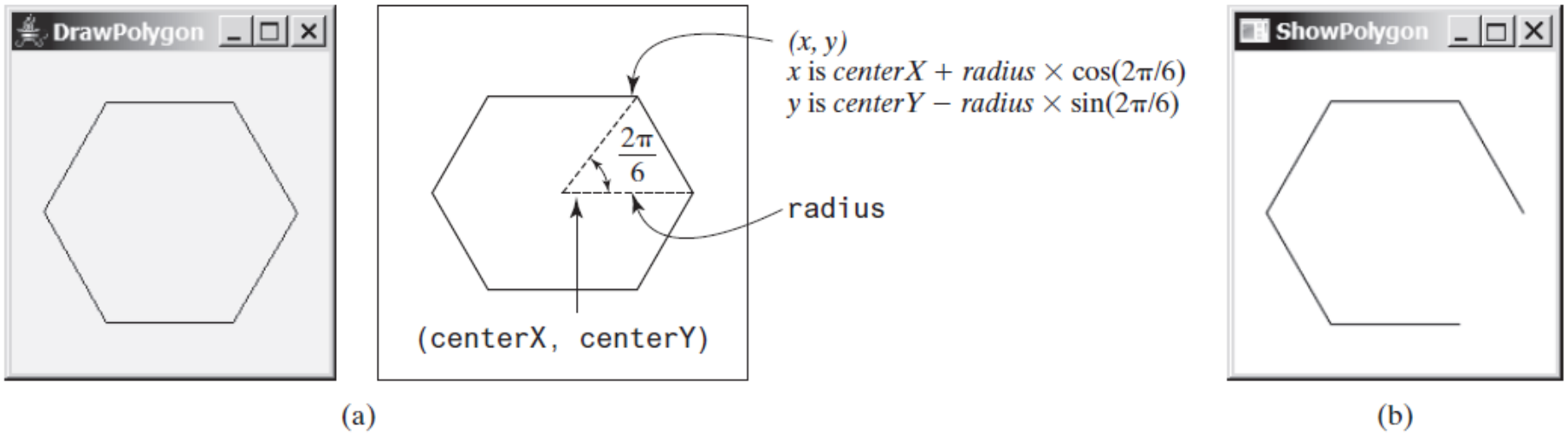


(b) Polyline

`Polygon` is closed and `Polyline` is not closed.

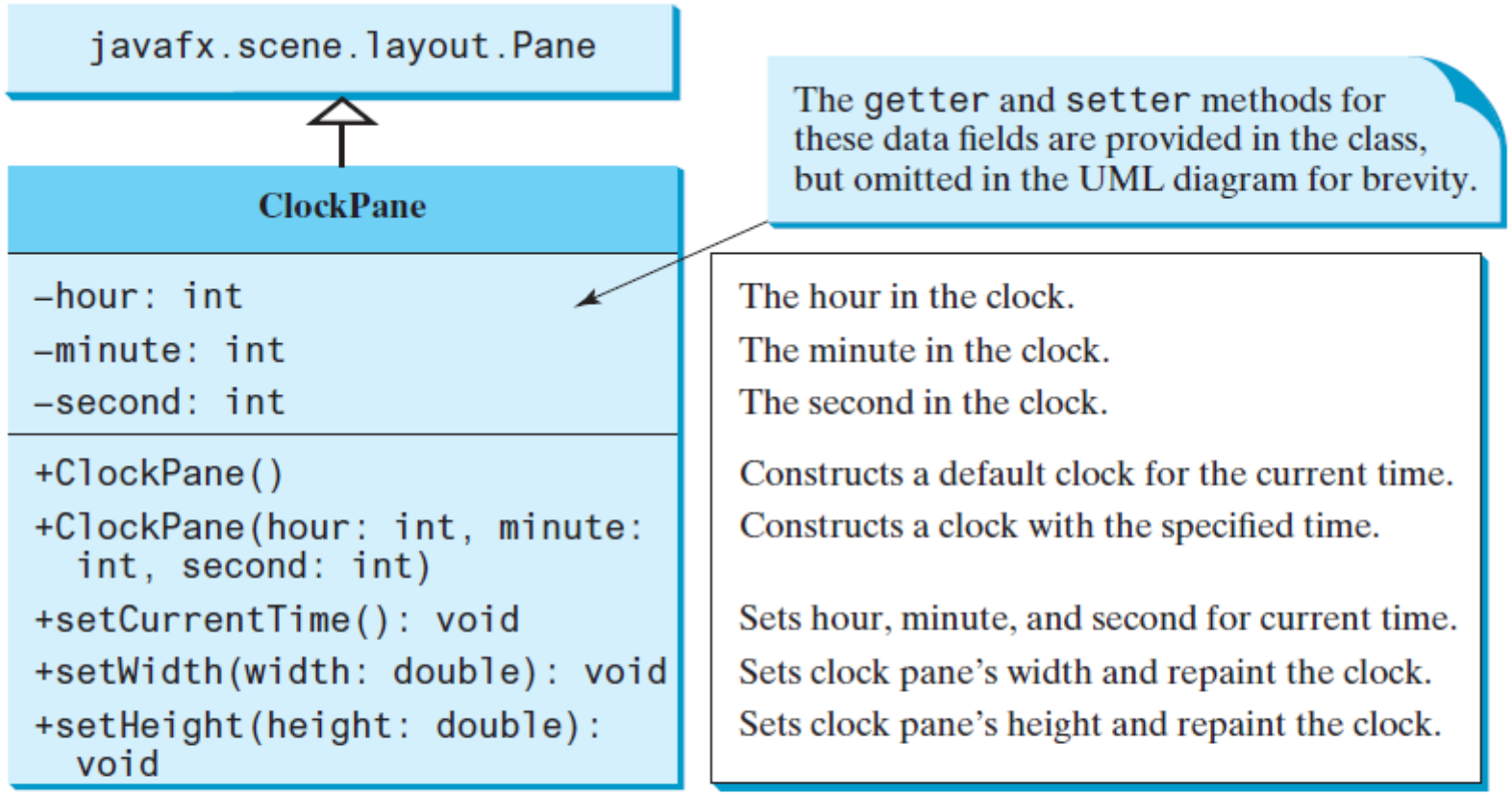
javafx.scene.shape.Polygon	
+Polygon()	Creates an empty Polygon.
+Polygon(double... points)	Creates a Polygon with the given points.
+getPoints(): ObservableList<Double>	Returns a list of double values as *x*- and *y*-coordinates of the points.

The `Polygon` class defines a polygon.



(a) A `Polygon` is displayed. (b) A `Polyline` is displayed.

If you replace `Polygon` by `Polyline` (line 23), the program displays a polyline as shown in Figure 14.40b. The `Polyline` class is used in the same way as `Polygon`, except that the starting and ending points are not connected in `Polyline`.



`ClockPane` displays an analog clock.

Assume `ClockPane` is available; we write a test program in Listing 14.20 to display an analog clock and use a label to display the hour, minute, and second, as shown in Figure 14.42.

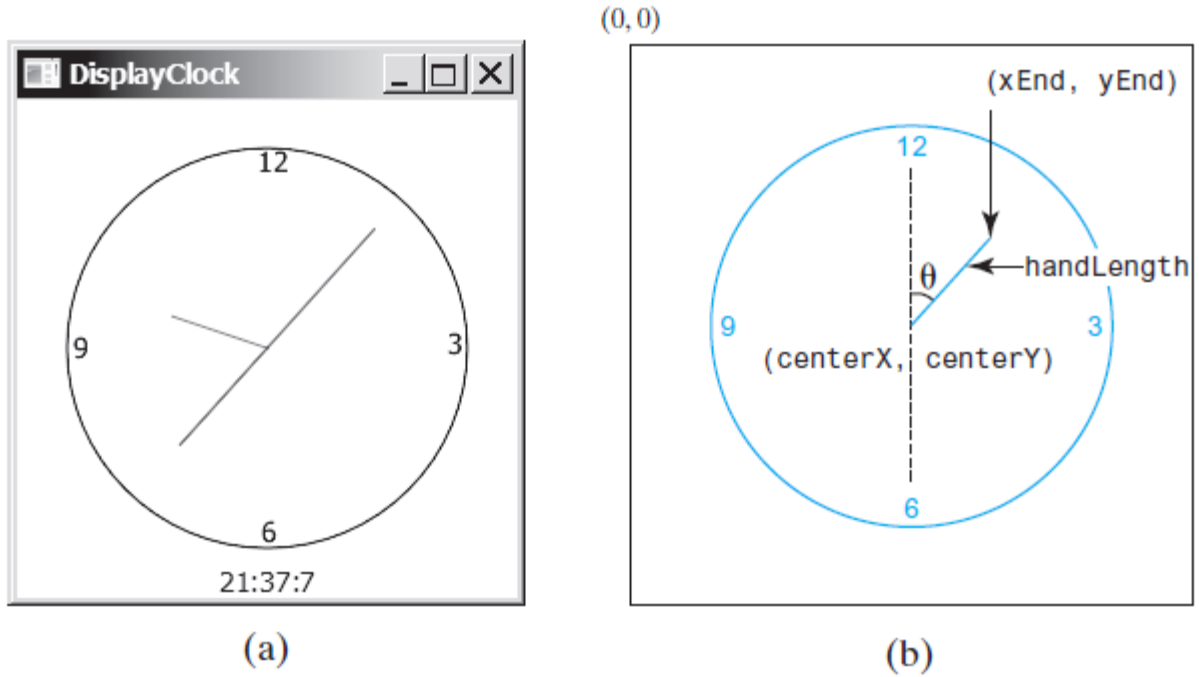


FIGURE 14.42 (a) The `DisplayClock` program displays a clock that shows the current time. (b) The endpoint of a clock hand can be determined, given the spanning angle, the hand length, and the center point.

DisplayClock.java

```
1  import javafx.application.Application;
2  import javafx.geometry.Pos;
3  import javafx.stage.Stage;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Label;
6  import javafx.scene.layout.BorderPane;
7
8  public class DisplayClock extends Application {
9    @Override // Override the start method in the Application class
10   public void start(Stage primaryStage) {
11     // Create a clock and a label
12     ClockPane clock = new ClockPane();                          create a clock
13     String timeString = clock.getHour() + ":" + clock.getMinute()
14       + ":" + clock.getSecond();
15     Label lblCurrentTime = new Label(timeString);               create a label
16
17     // Place clock and label in border pane
18     BorderPane pane = new BorderPane();
19     pane.setCenter(clock);                                      add a clock
20     pane.setBottom(lblCurrentTime);                             add a label
21     BorderPane.setAlignment(lblCurrentTime, Pos.TOP_CENTER);
22
23     // Create a scene and place it in the stage
24     Scene scene = new Scene(pane, 250, 250);
25     primaryStage.setTitle("DisplayClock"); // Set the stage title
26     primaryStage.setScene(scene); // Place the scene in the stage
27     primaryStage.show(); // Display the stage
28   }
29 }
```

Event Listeners

Similar to Exceptions: “bubbles up”

Event Handlers are like “catch”

There are different rules for the system to find the target widget that has the focus for a particular events. EG cursor position for mouse events, button presses, etc. Consider issues such as if an element is hidden by another it's the one on top and it is considered to be the target.

Events and Change Listeners

1. Determine the Event Target

EventTarget interface

- Window
- Scene
- Node

Events and Change Listeners

.setOn SomeAction**()** methods for nodes:

“KeyPressed”
“KeyReleased”
“KeyTyped”
“MouseClicked”
“MouseExited”
many more...

Events and Change Listeners

.setOnAction() method for

- Buttons
- RadioButtons
- CheckBoxes

Events and Change Listeners

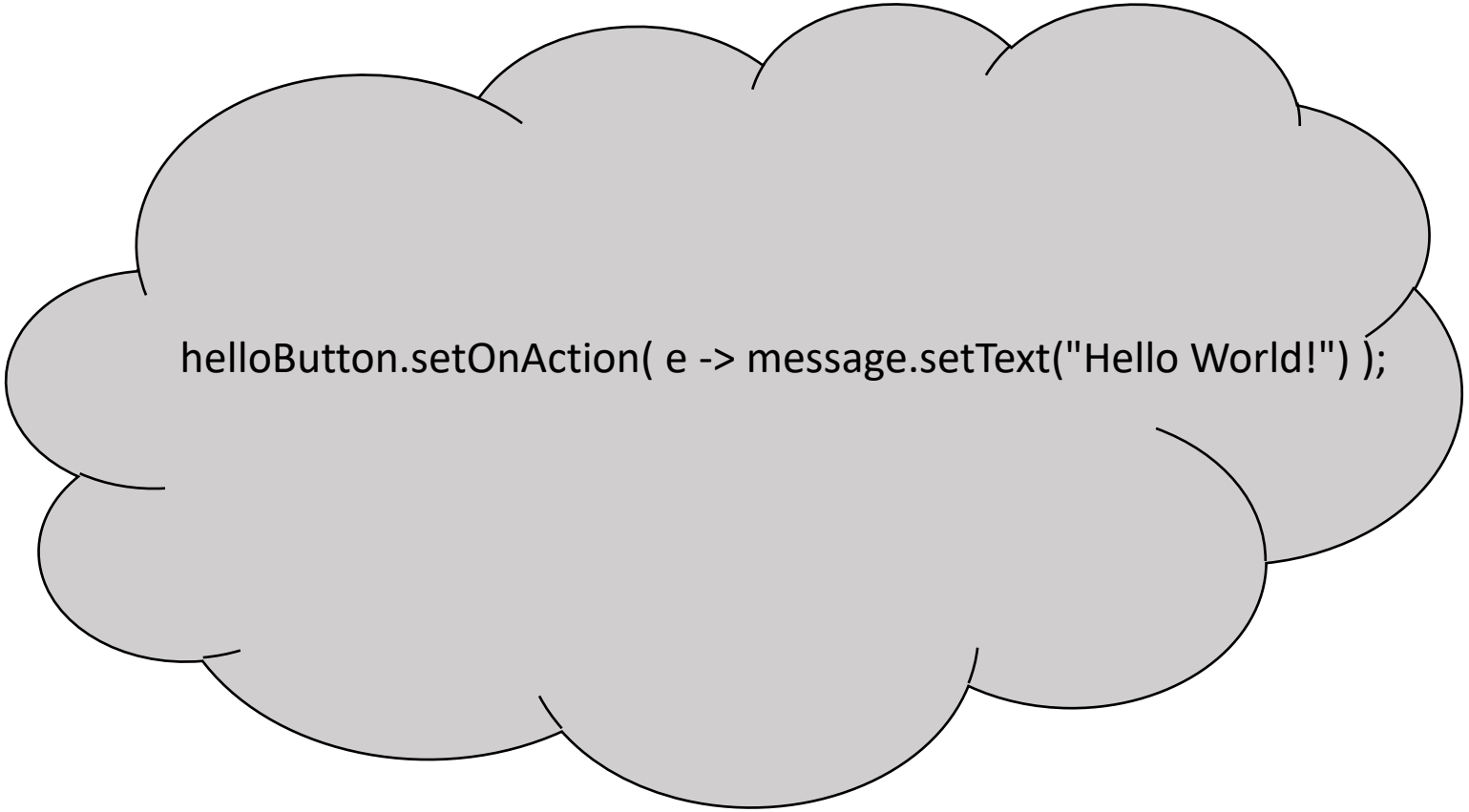
Use Lambda Expressions:

```
Button btn = new Button();
btn.setText("Say 'Hi'");
btn.setOnAction(e -> System.out.println("Hi!") );
```

```
public class HelloWorldFX extends Application {

    public void start(Stage stage) {

        Label message = new Label("First FX Application!");
        message.setFont( new Font(40) );

        Button helloButton = new Button("Say Hello");
        helloButton.setOnAction( e -> message.setText("Hello World!") );
        Button goodbyeButton = new Button("Say Goodbye");
        goodbyeButton.setOnAction( e -> message.setText("Goodbye!!") );
        Button quitButton = new Button("Quit");
        quitButton.setOnAction( e -> Platform.exit() );

        HBox buttonBar = new HBox( 20, helloButton, goodbyeButton, quitButton );
        buttonBar.setAlignment(Pos.CENTER);
        BorderPane root = new BorderPane();
        root.setCenter(message);
        root.setBottom(buttonBar);

        Scene scene = new Scene(root, 450, 200);
        stage.setScene(scene);
        stage.setTitle("JavaFX Test");
        stage.show();

    } // end start();

    public static void main(String[] args) {
        launch(args);  // Run this Application.
    }

} // end class HelloWorldFX
```

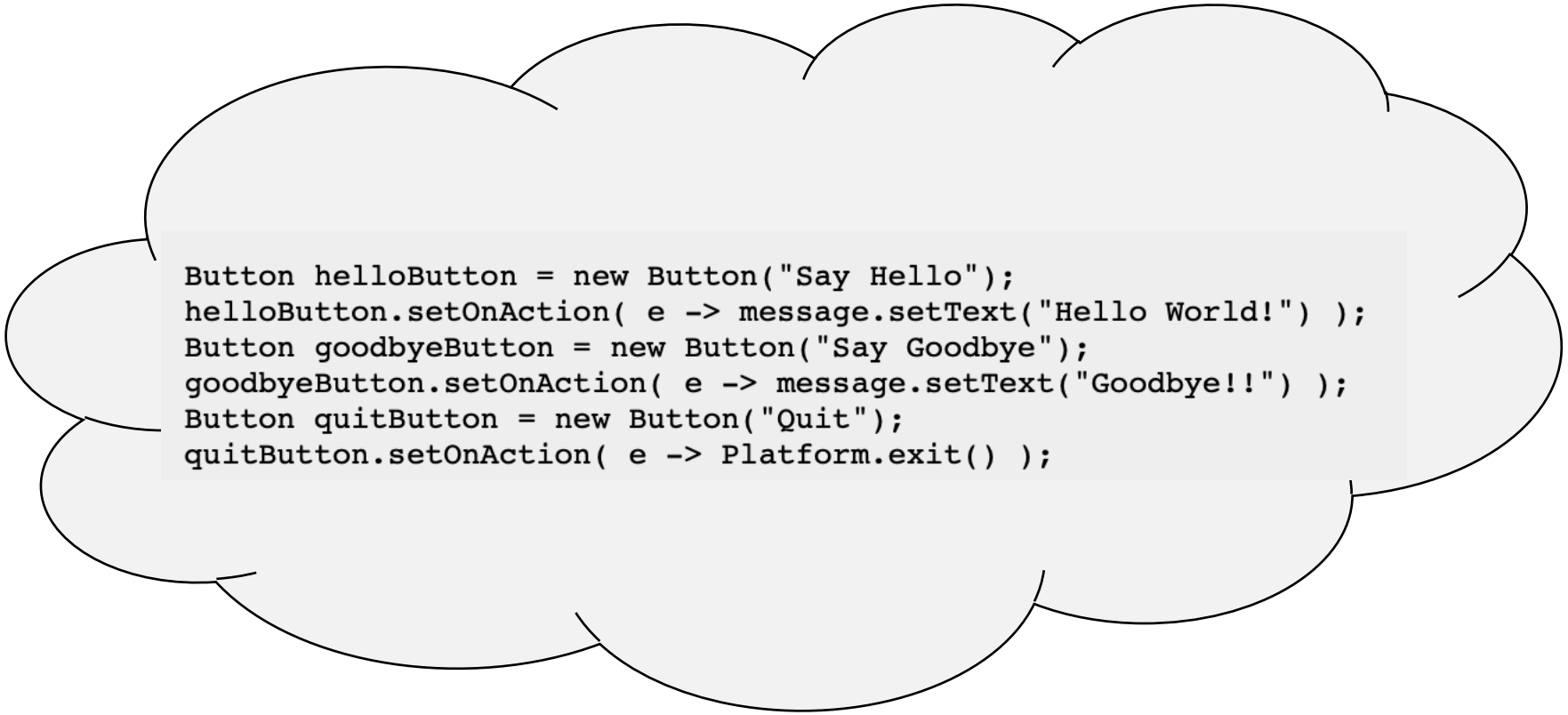
Events and Change Listeners

Use Lambda Expressions:



Events and Change Listeners

Use Lambda Expressions:



```
Button helloButton = new Button("Say Hello");
helloButton.setOnAction( e -> message.setText("Hello World!") );
Button goodbyeButton = new Button("Say Goodbye");
goodbyeButton.setOnAction( e -> message.setText("Goodbye!!") );
Button quitButton = new Button("Quit");
quitButton.setOnAction( e -> Platform.exit() );
```

Exercise

- Download the HelloWorldFX Application that was discussed in this presentation from blackboard resources section or the Java8 online text
- Run the program in your IDE
- Test some changes and familiarize yourself with the methods used (e.g. change the text in the action listener, remove the the quit button action, etc)

You can also define Event Filters to intercept (block) some events or override them

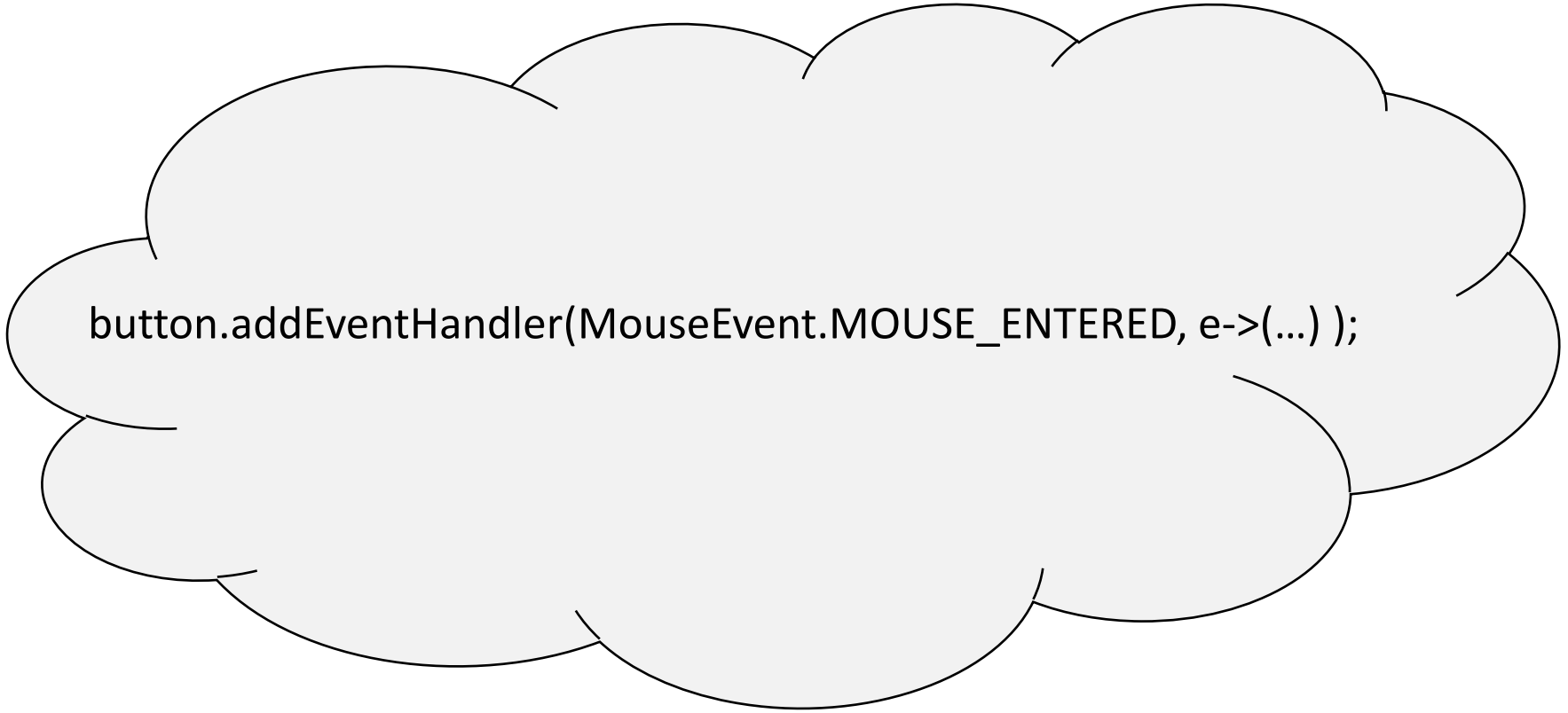
Event Filters are activated before event handlers.

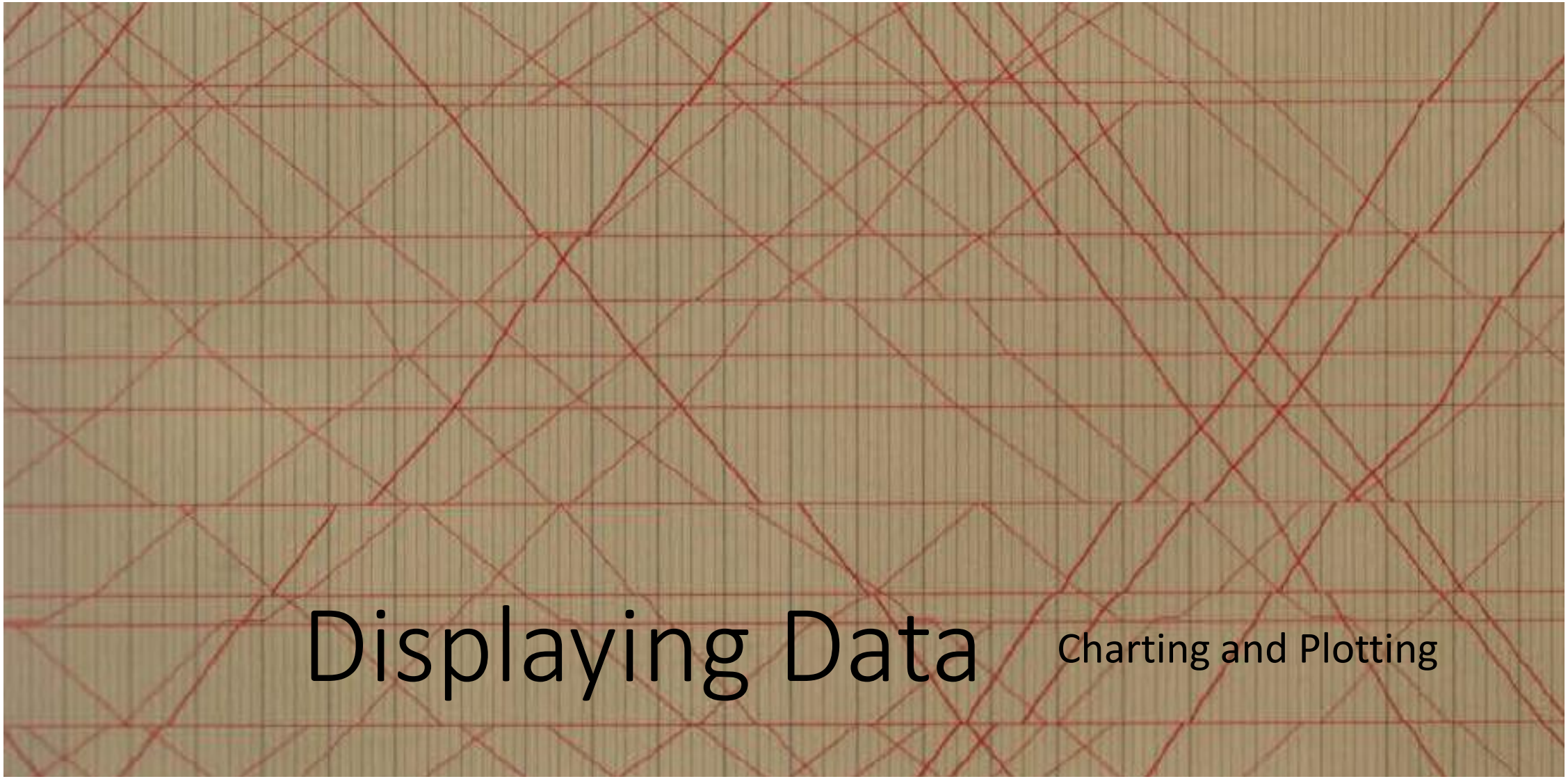
When an event occurs it is first passed through the chain to the target (for example if you have a scene with a button, or even a shape) the event would normally bubble up from the target through a pane -> scene -> stage and be acted on by an event handler if there are any.

Before the event is passed to **any** event handlers, it is first passed to any event filters that exist in the pane, scene, stage or target would be applied first and could prevent the child nodes from receiving the event (block) and possibly do an alternative action (override).

Events and Change Listeners

Instead of .setOnxxxx() methods, you can use addEventHandler():



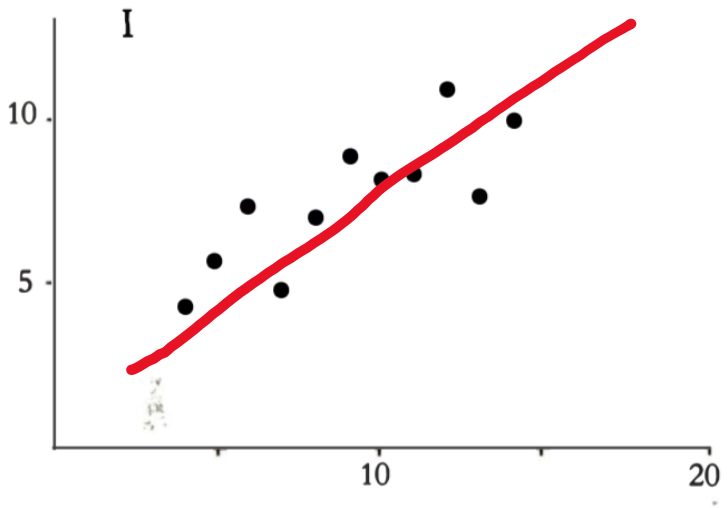

Displaying Data
Charting and Plotting

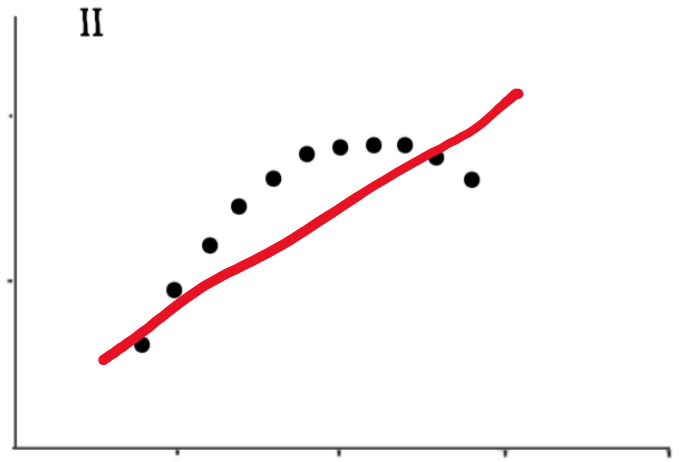
“Graphical excellence consists of the efficient communication of complex quantitative ideas.”

I		II		III		IV	
X	Y	X	Y	X	Y	X	Y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

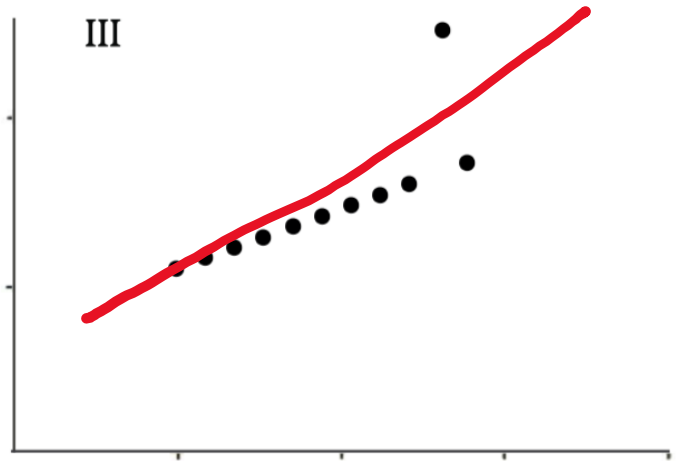
→

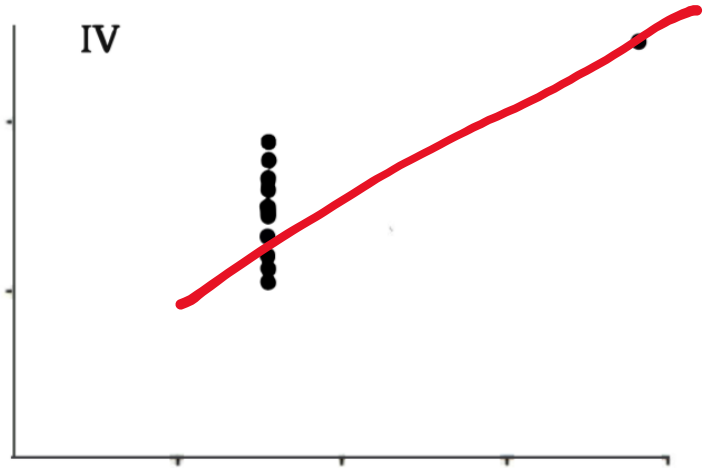
$N = 11$
mean of X's $= 9.0$
mean of Y's $= 7.5$
equation of regression line: $Y = 3 + 0.5X$
standard error of estimate of slope $= 0.118$
$t = 4.24$
sum of squares $X - \overline{X} = 110.0$
regression sum of squares $= 27.50$
residual sum of squares of $Y = 13.75$
correlation coefficient $= .82$
$r^2 = .67$

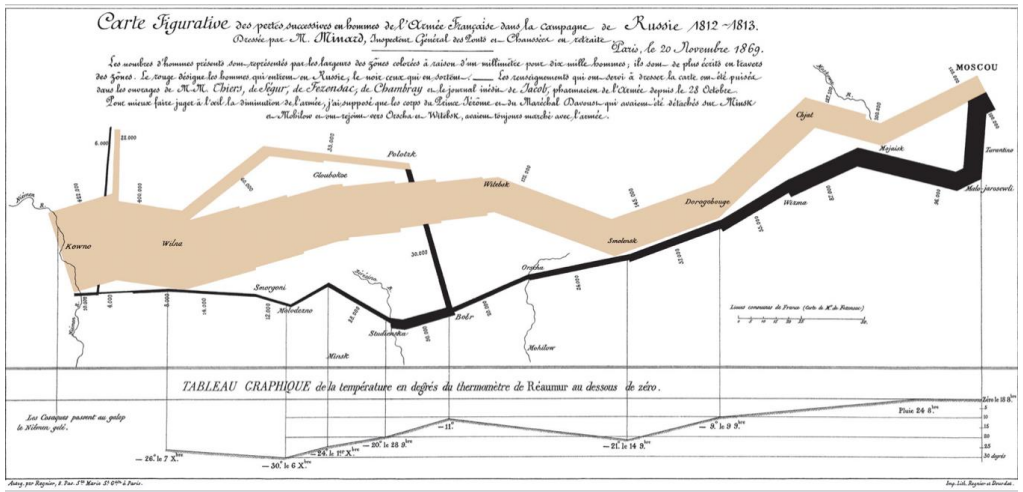




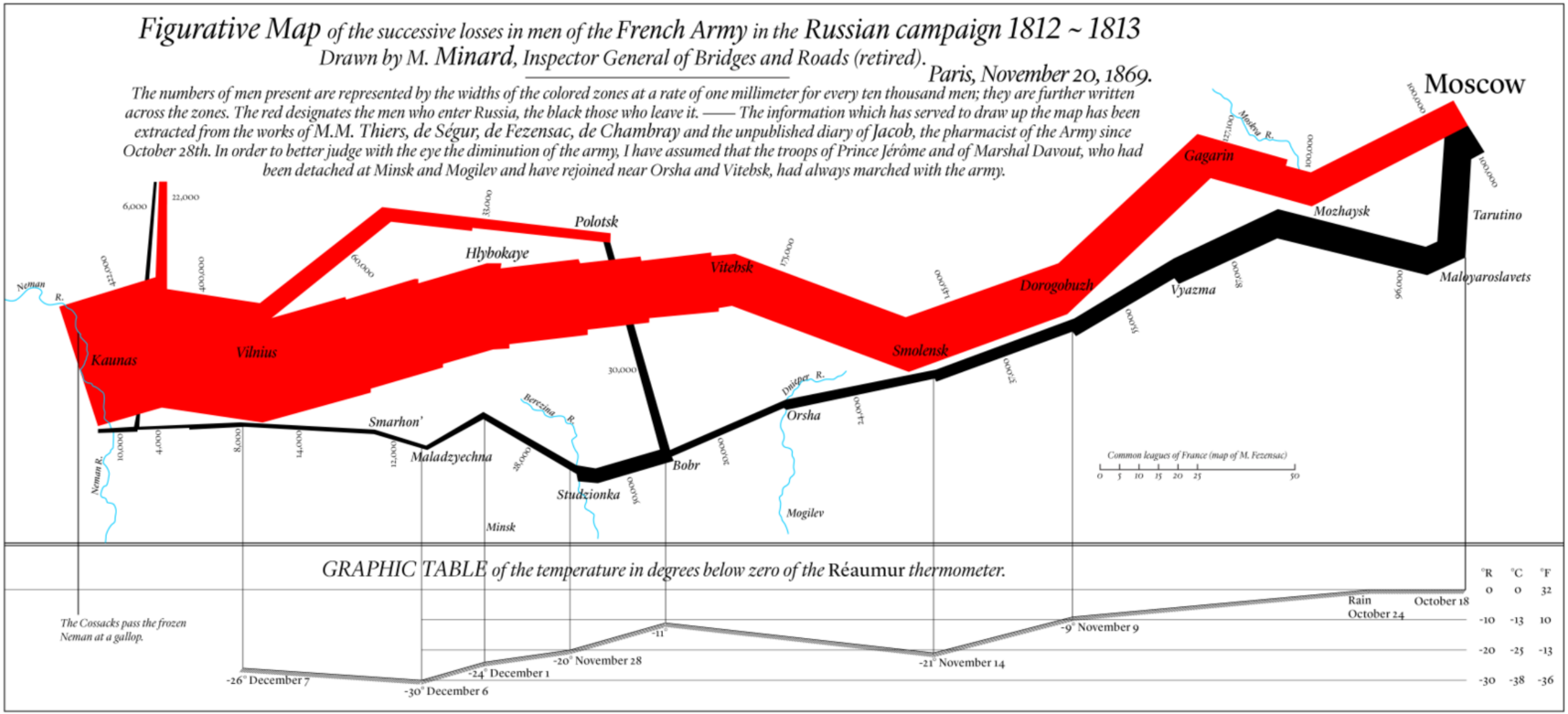
$$Y = 3 + 0.5X$$



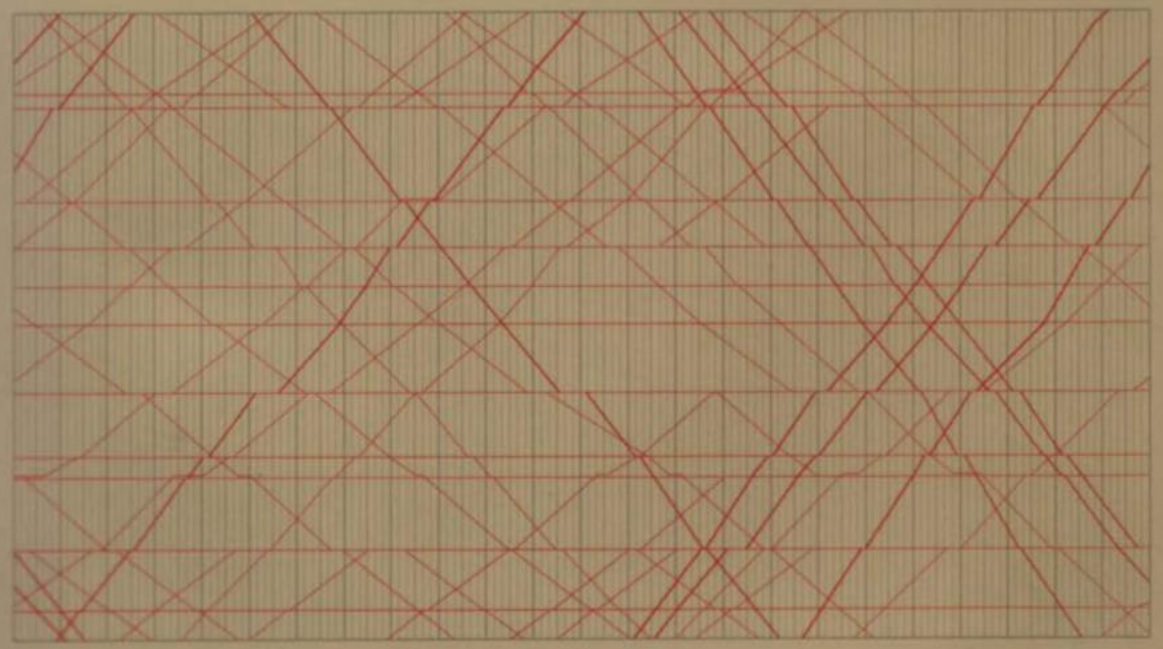




"The best statistical graphic ever drawn", is how statistician Edward **Tufte** described this



(This is a modern rendering in English – the original was in French)



SECOND EDITION

The Visual Display of Quantitative Information

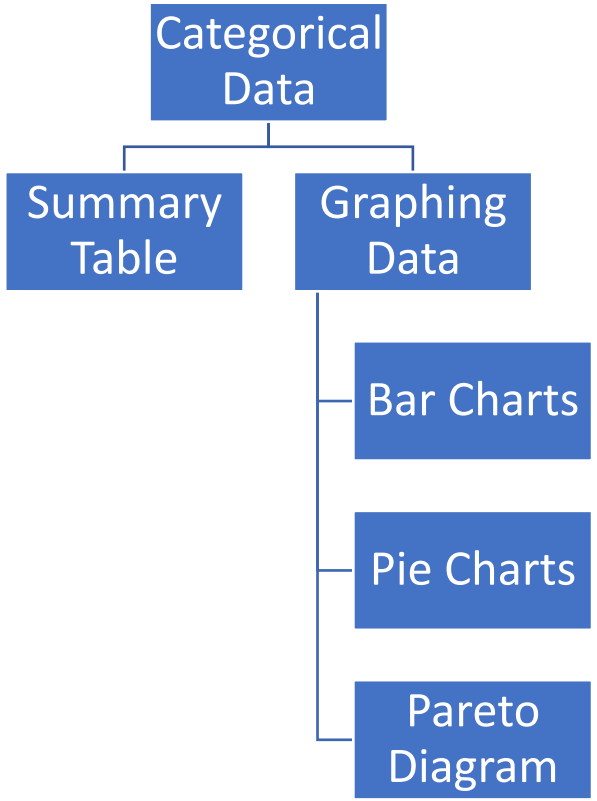
EDWARD R. TUFTE

Principles for Data Presentation

- Communicates complex ideas with clarity, precision and efficiency
- Gives the largest number of ideas in the most efficient manner
- Almost always uses several dimensions
- Requires telling the truth about the data

- ”Data Ink Ratio”

Tabulating and Graphing Categorical Data

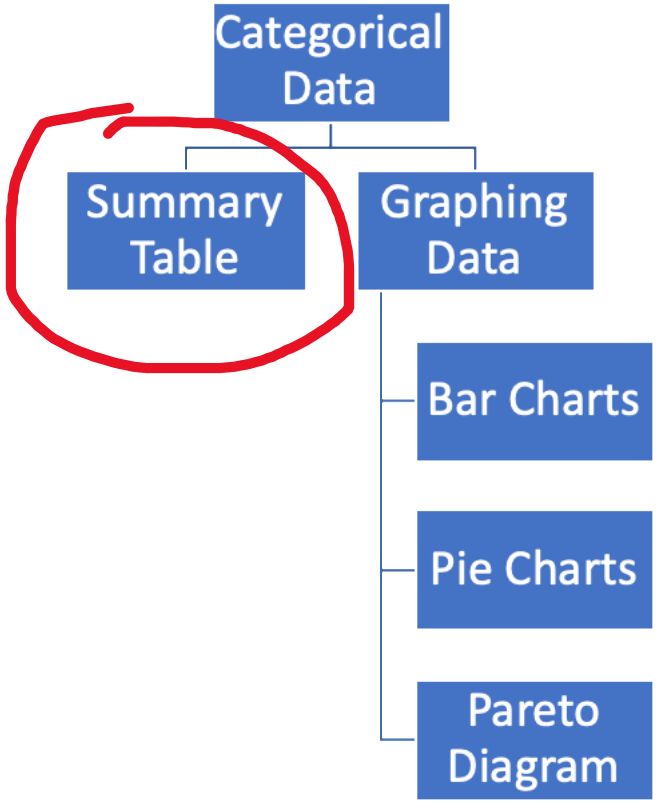


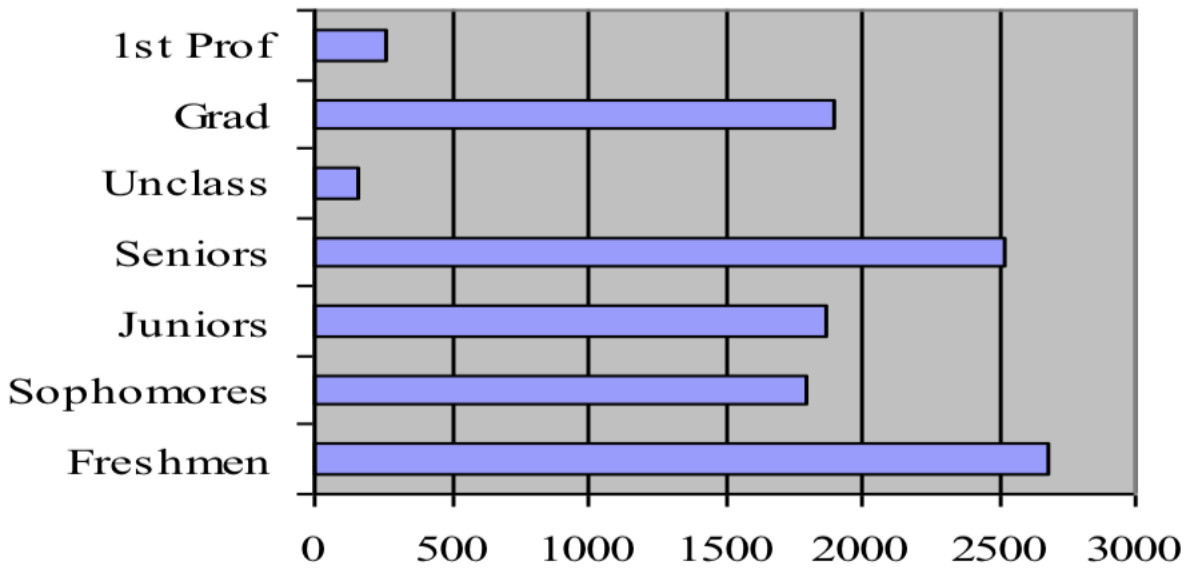
https://pdfs.semanticscholar.org/1d72/1668009bc65c899a6a631686427d987806d4.pdf?_ga=2.50376045.557616602.1585118213-775591735.1585118213

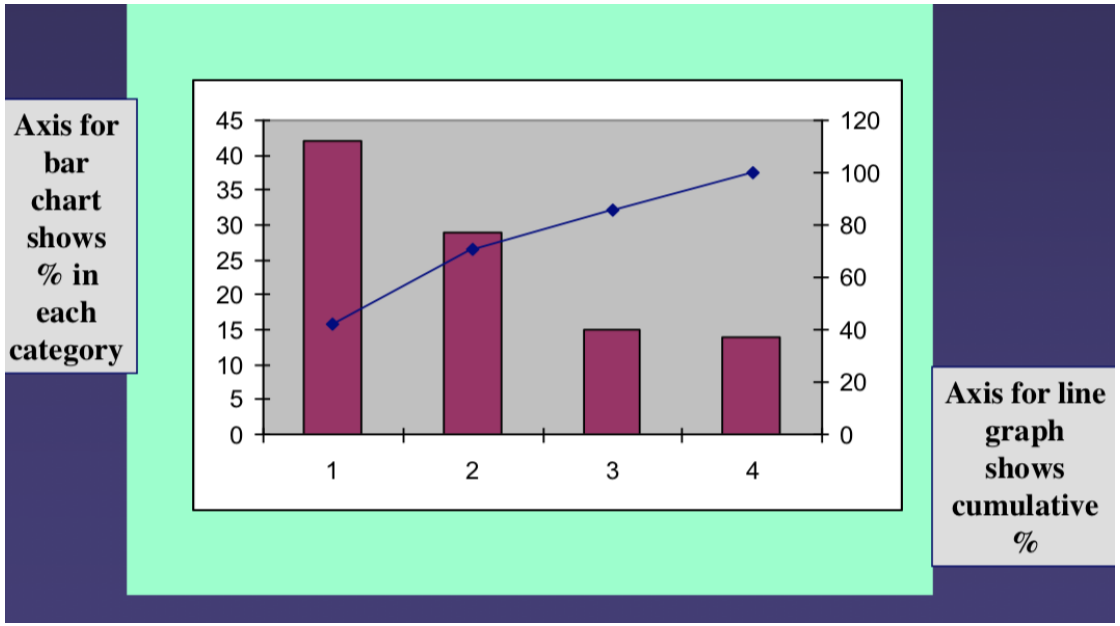
Summary Table of University Revenues

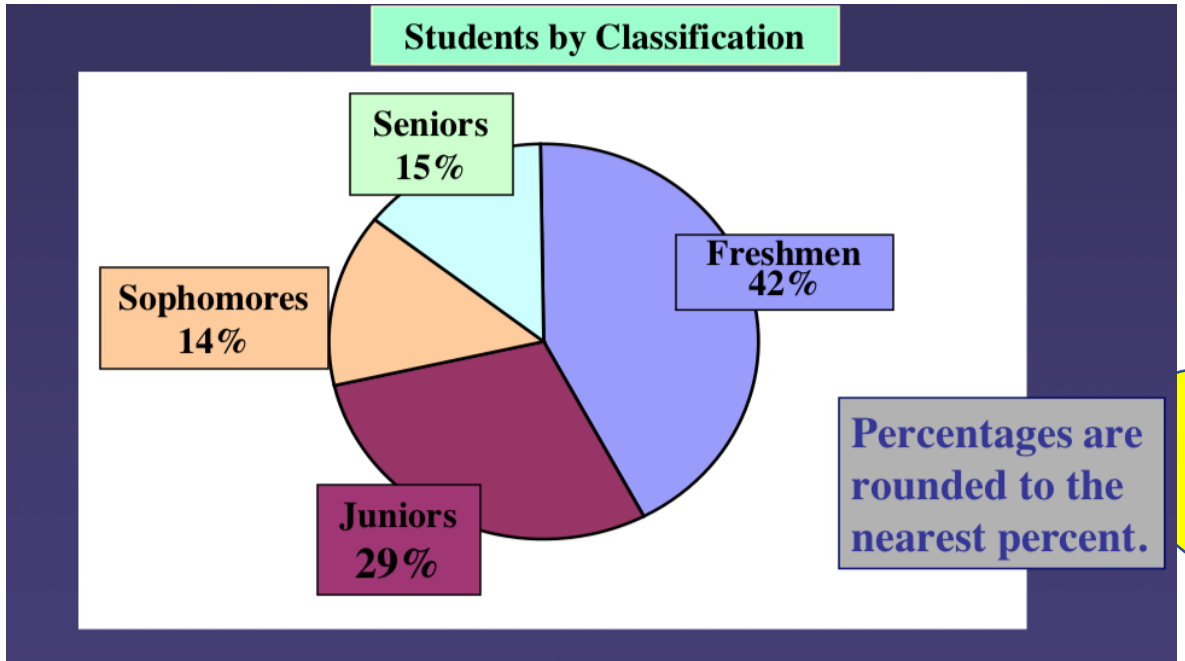
Revenue Category	Amount (in thousands $)	Percentage
Patient Services	46.5	42.27
Tution fees	32	29.09
Appropriations	15.5	14.09
Grants/Contracts	16	14.55
Total	**110**	**100**

Variables are categorical

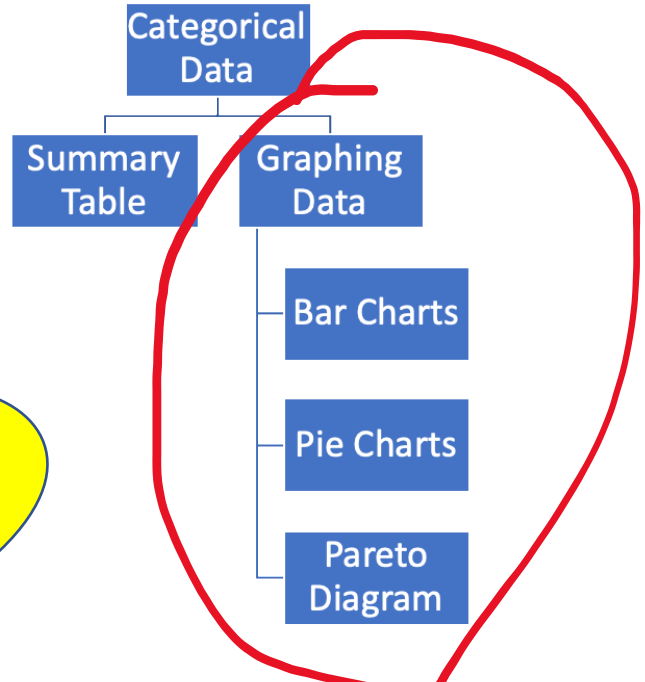


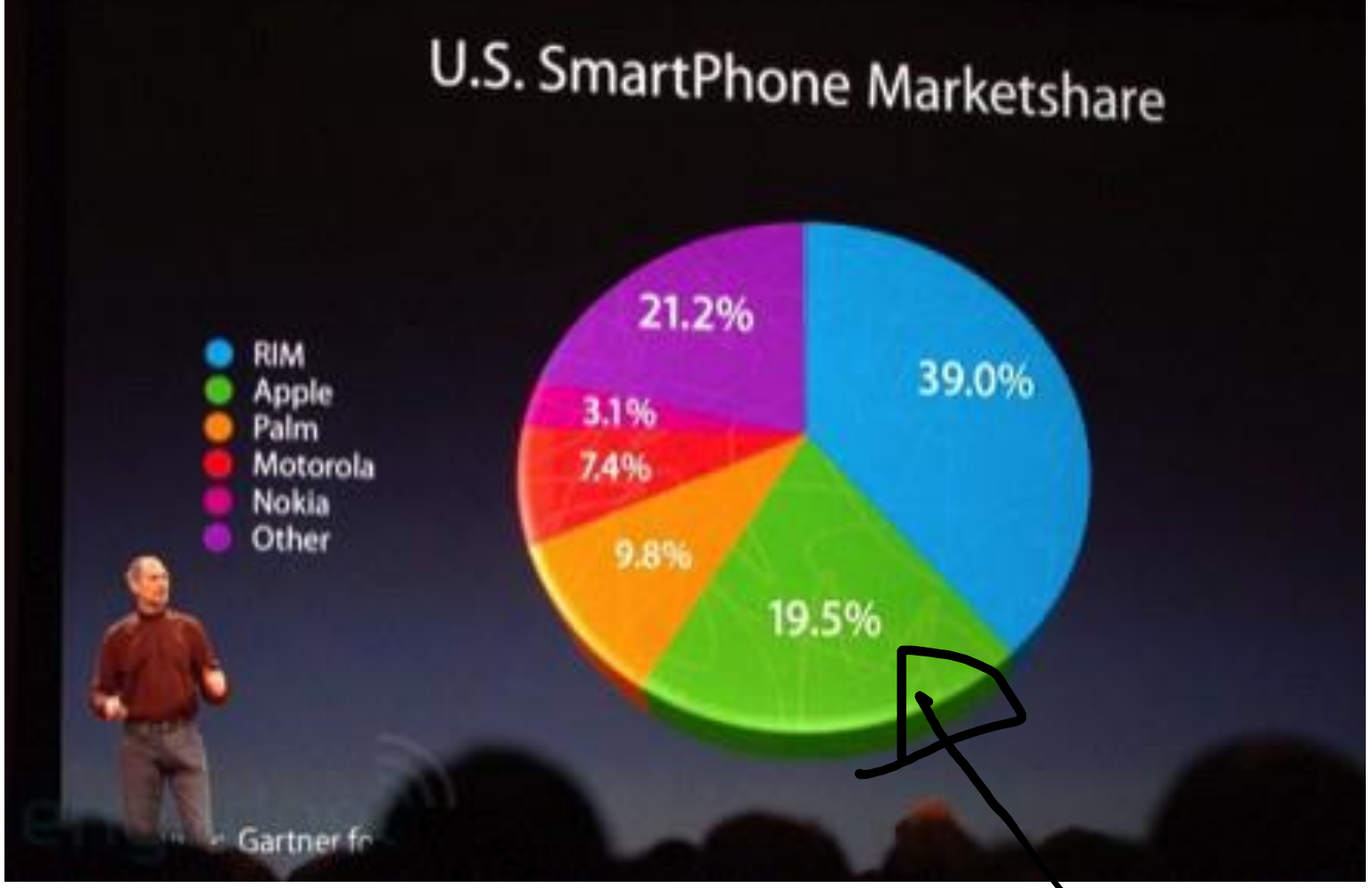






Pie charts are not liked by Tufte





Looks bigger because of perspective

Common Charting Errors

- Using ‘chart junk’
- No relative basis
  In comparing data
- Batches
- Compressing the Vertical axis
- No zero point on the Vertical axis

Bad Presentation

Minimum Wage


1960: $1.00

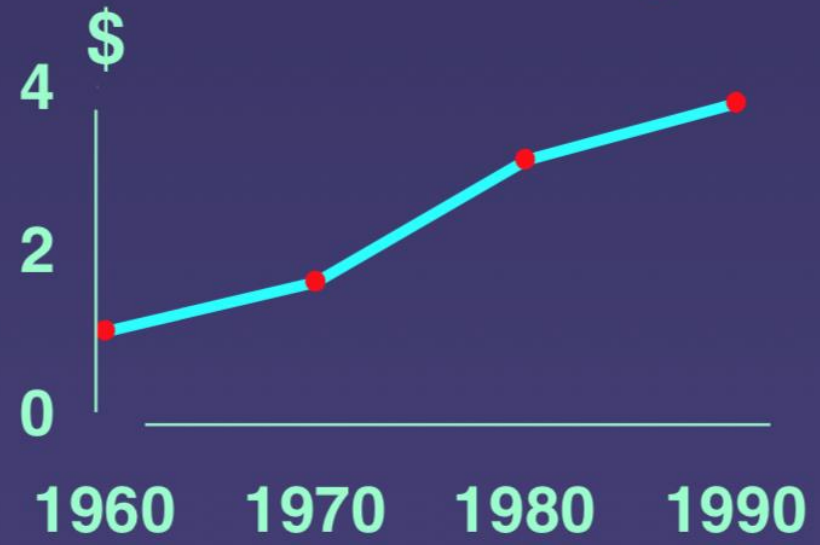

1970: $1.60

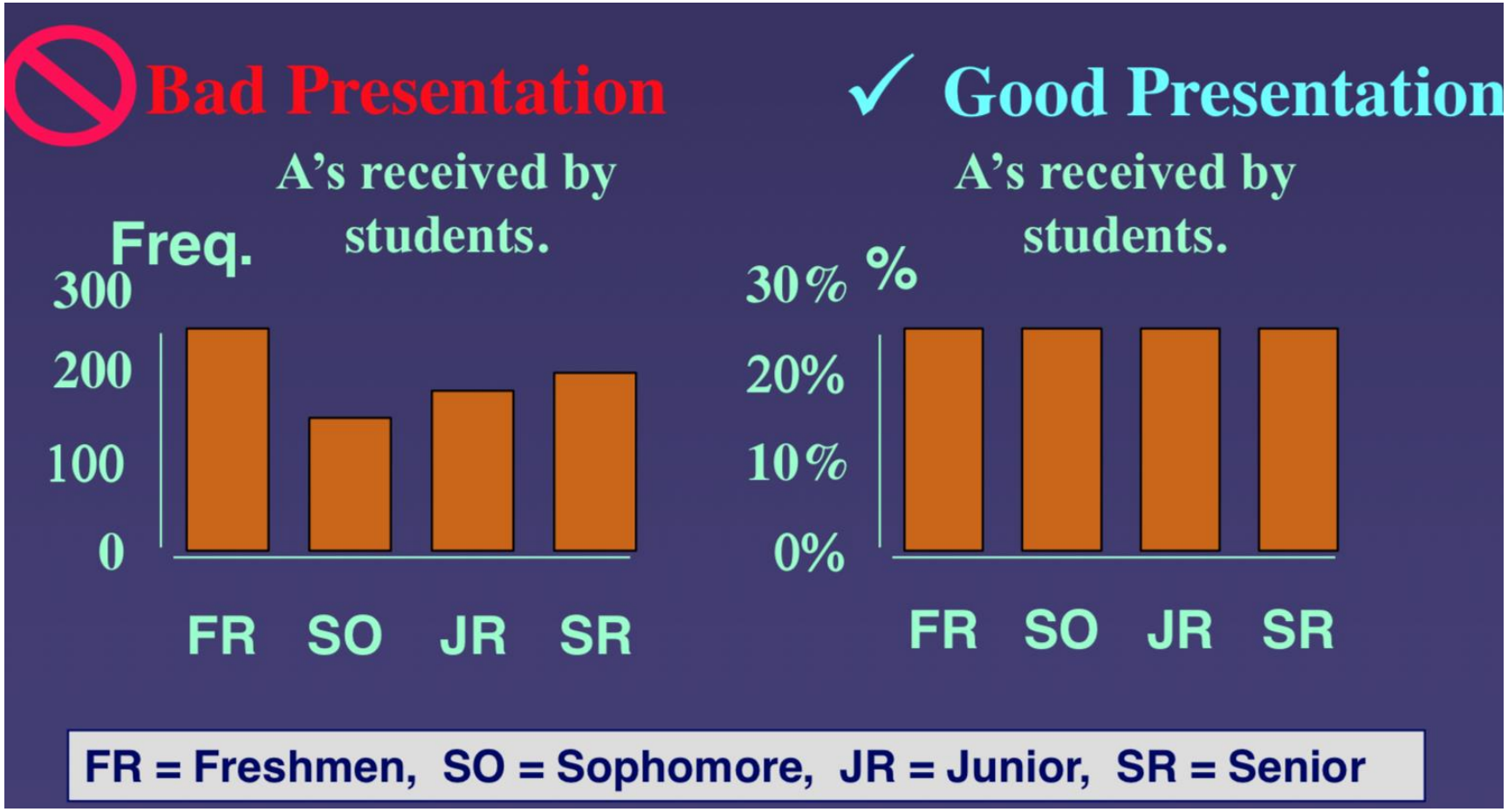
1980: $3.10

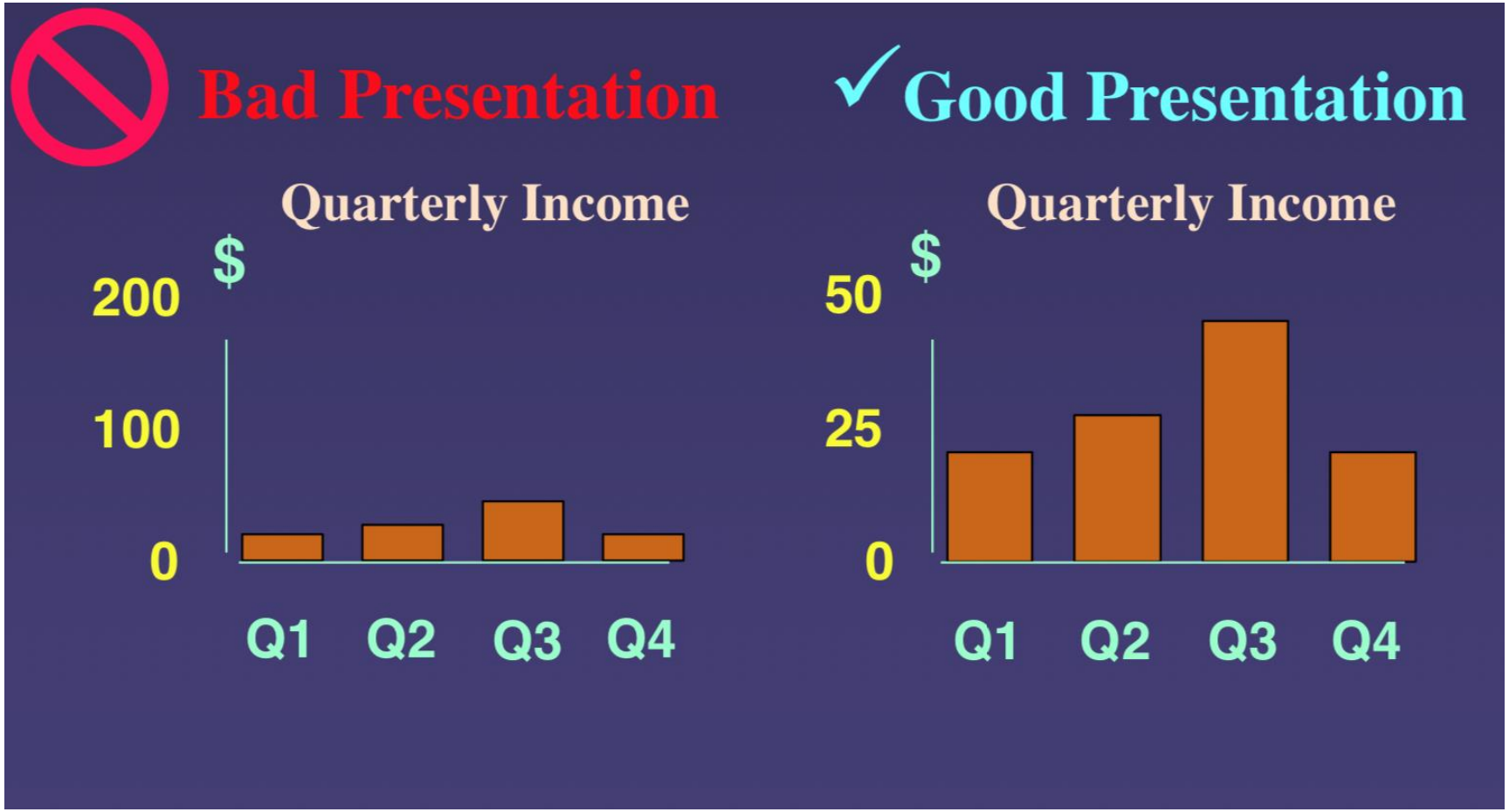
1990: $3.80

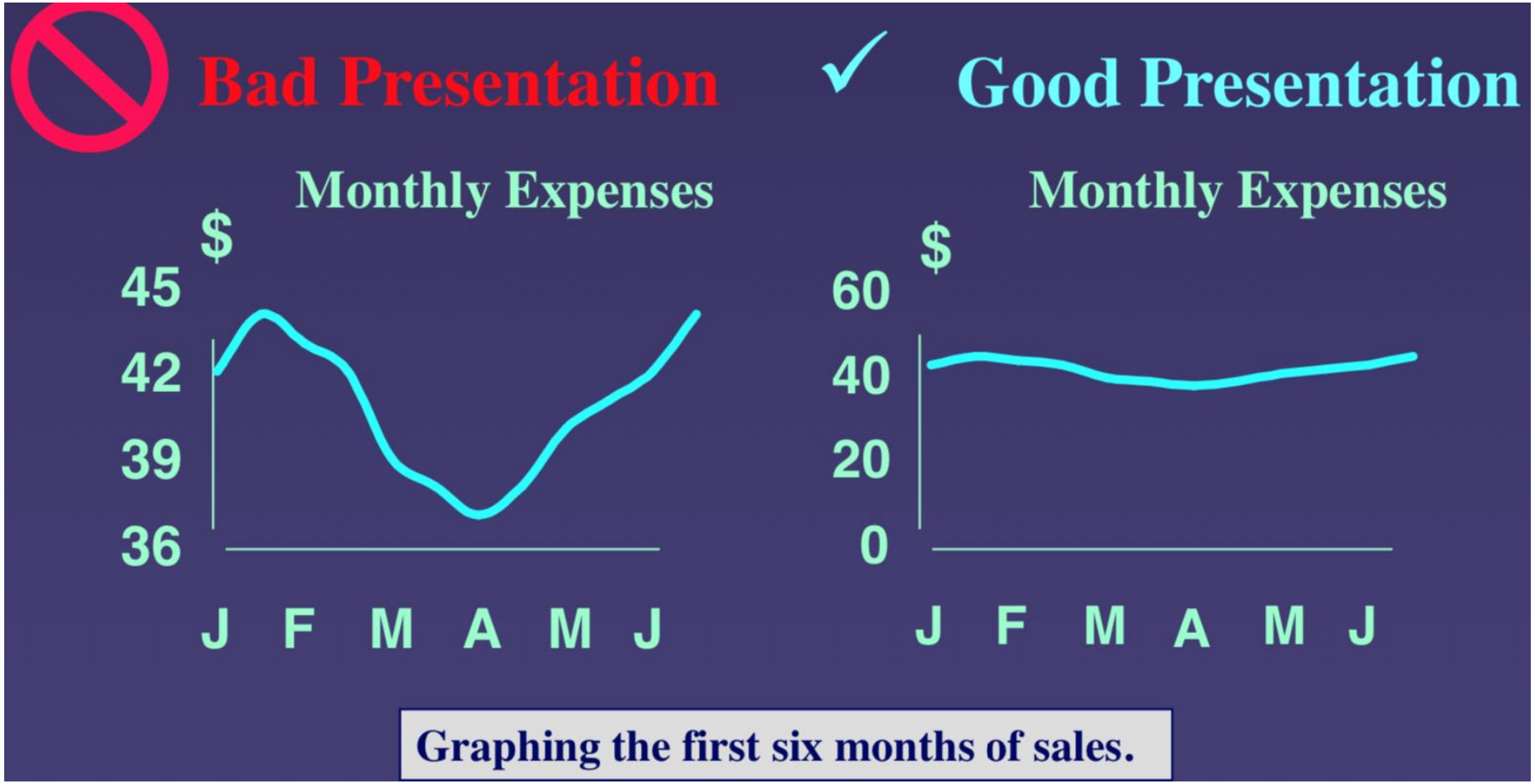
Good Presentation

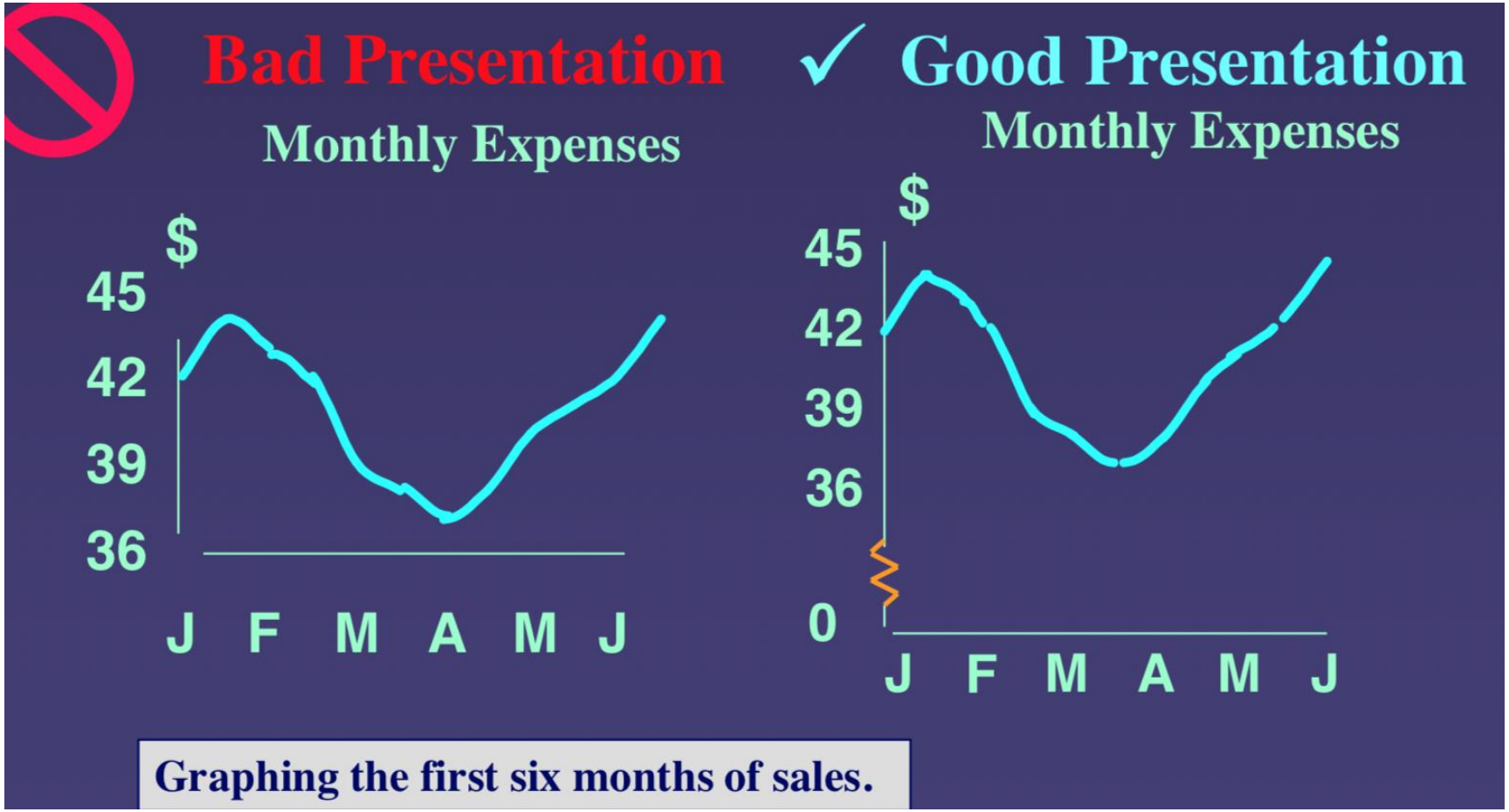
Minimum Wage




Bad Presentation
A's received by students.
Freq.
300
200
100
0
FR SO JR SR
Good Presentation
A's received by students.
%
30%
20%
10%
0%
FR SO JR SR
FR = Freshmen, SO = Sophomore, JR = Junior, SR = Senior


Bad Presentation
Quarterly Income
$
200
100
0
Q1
Q2
Q3
Q4
Good Presentation
Quarterly Income
$
50
25
0
Q1
Q2
Q3
Q4


Bad Presentation
Monthly Expenses
$
45
42
39
36
J F M A M J
Good Presentation
Monthly Expenses
$
60
40
20
0
J F M A M J
Graphing the first six months of sales.


Bad Presentation
Monthly Expenses
$
45
42
39
36
J F M A M J
✓ Good Presentation
Monthly Expenses
$
45
42
39
36
0
J F M A M J
Graphing the first six months of sales.

Online Tutorials on JavaFX:

JavaFX教程 @ 易百教程
https://www.yiibai.com/javafx

JavaFX – Charts @ oracle.com
https://docs.oracle.com/javafx/2/api/javafx/scene/chart/package-summary.html

JavaFX – Charts @ tutorialspoint.com
http://www.tutorialspoint.com/javafx/javafx_charts.htm

Visualizing Data with Plots

Data visualization is an important and exciting component in Scientific and Engineering Domains, especially in data science.

Data visualization should always take into consideration the audience, there are roughly three kinds of consumers of a visualization:
yourself, industry expert, everybody else

Visualizing Data for Yourself

The first is yourself, the all-knowing expert who is most likely iterating quickly on an analysis or algorithm development.

Your requirements are to see the data as plainly and quickly as possible. Things such as setting plot titles, axis labels, smoothing, legends, or date formatting might not be important, because you are intimately aware of what you are looking at.

In essence, we often plot data to get a quick overview of the data landscape, without concerning ourselves with how others will view it.

Visualizing Data for Industry Expert

The second consumer of data visualizations is the industry expert. After you have solved a data science problem and you think it's ready to share, it's essential to fully label the axis, put a meaningful, descriptive title on it, make sure any series of data are described by a legend, and ensure that the graphic you have created can mostly tell a story on its own.

Even if it's not visually stunning, your colleagues and peers will probably not be concerned with eye candy, but rather the message you are trying to convey.

In fact, it will be much easier to make a scientific evaluation on the merits of the work if the visualization is clear of graphical widgets and effects. Of course, this format is also essential for archiving your data. One month later, you will not remember what those axes are if you don't label them now!

Visualizing Data for Everybody

The third category of visualization consumer is everybody else. This is the time to get creative and artistic, because a careful choice of colors and styles can make good data seem great.

Be cautious, however, of the tremendous amount of time and effort you will spend preparing graphics at this level of consumer.

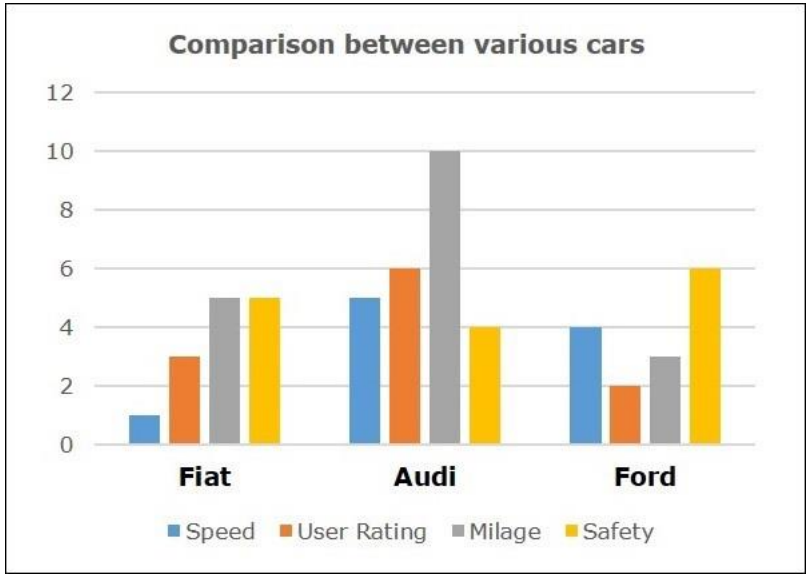
An added advantage of using JavaFX is the interactivity allowed via mouse options. This enables you to build a graphical application similar to many of the web-based dashboards you are accustomed to.

```
import javafx.scene.chart.*;
```

class PieChart

class XYChart
x can be
numeric or
a category
name

- AreaChart
- BarChart
- BubbleChart
- LineChart
- ScatterChart
- StackedAreaChart
- StackedBarChart



A bar chart

https://docs.oracle.com/javafx/2/api/javafx/scene/chart/package-summary.html

Package javafx.scene.chart

The JavaFX User Interface provides a set of chart components that are a very convenient way for data visualization.

See: Description

Class Summary

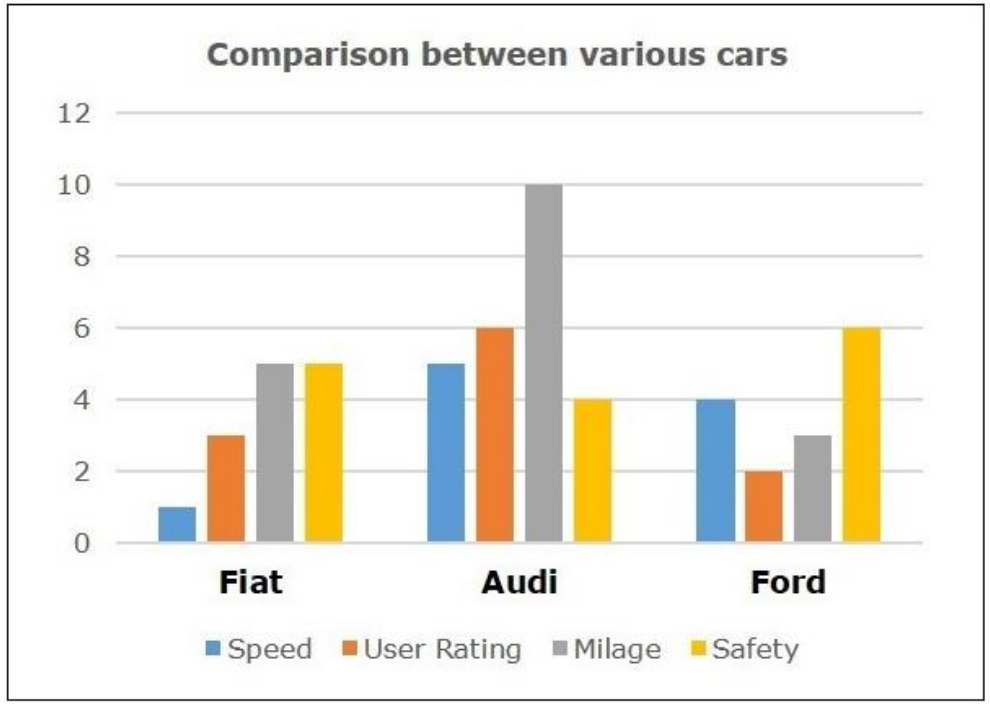
Class	Description
AreaChart<X,Y>	AreaChart - Plots the area between the line that connects the data points and the 0 line on the Y axis.
AreaChartBuilder<X,Y,B extends **AreaChartBuilder**<X,Y,B>>	Builder class for javafx.scene.chart.AreaChart
Axis<T>	Base class for all axes in JavaFX that represents an axis drawn on a chart area.
Axis.TickMark<T>	TickMark represents the label text, its associated properties for each tick along the Axis.
AxisBuilder<T,B extends **AxisBuilder**<T,B>>	Builder class for javafx.scene.chart.Axis
BarChart<X,Y>	A chart that plots bars indicating data values for a category.
BarChartBuilder<X,Y,B extends **BarChartBuilder**<X,Y,B>>	Builder class for javafx.scene.chart.BarChart
BubbleChart<X,Y>	Chart type that plots bubbles for the data points in a series.
BubbleChartBuilder<X,Y,B extends **BubbleChartBuilder**<X,Y,B>>	Builder class for javafx.scene.chart.BubbleChart
CategoryAxis	A axis implementation that will works on string categories where each value as a unique category(tick mark) along the axis.
CategoryAxisBuilder	Builder class for javafx.scene.chart.CategoryAxis
Chart	Base class for all charts.
ChartBuilder<B extends **ChartBuilder**<B>>	Builder class for javafx.scene.chart.Chart

LineChart<X,Y>	Line Chart plots a line connecting the data points in a series.
LineChartBuilder<X,Y,B extends **LineChartBuilder**<X,Y,B>>	Builder class for javafx.scene.chart.LineChart
NumberAxis	A axis class that plots a range of numbers with major tick marks every "tickUnit".
NumberAxis.DefaultFormatter	Default number formatter for NumberAxis, this stays in sync with auto-ranging and formats values appropriately.
NumberAxisBuilder	Builder class for javafx.scene.chart.NumberAxis
PieChart	Displays a PieChart.
PieChart.Data	PieChart Data Item, represents one slice in the PieChart
PieChartBuilder<B extends **PieChartBuilder**<B>>	Builder class for javafx.scene.chart.PieChart
ScatterChart<X,Y>	Chart type that plots symbols for the data points in a series.
ScatterChartBuilder<X,Y,B extends **ScatterChartBuilder**<X,Y,B>>	Builder class for javafx.scene.chart.ScatterChart
StackedAreaChart<X,Y>	StackedAreaChart is a variation of `AreaChart` that displays trends of the contribution of each value.
StackedAreaChartBuilder<X,Y,B extends **StackedAreaChartBuilder**<X,Y,B>>	Builder class for javafx.scene.chart.StackedAreaChart
StackedBarChart<X,Y>	StackedBarChart is a variation of `BarChart` that plots bars indicating data values for a category.
StackedBarChartBuilder<X,Y,B extends **StackedBarChartBuilder**<X,Y,B>>	Builder class for javafx.scene.chart.StackedBarChart
ValueAxis<T extends java.lang.Number>	A axis who's data is defined as Numbers.
ValueAxisBuilder<T extends java.lang.Number,B extends **ValueAxisBuilder**<T,B>>	Builder class for javafx.scene.chart.ValueAxis
XYChart<X,Y>	Chart base class for all 2 axis charts.
XYChart.Data<X,Y>	A single data item with data for 2 axis charts
XYChart.Series<X,Y>	A named series of data items
XYChartBuilder<X,Y,B extends **XYChartBuilder**<X,Y,B>>	Builder class for javafx.scene.chart.XYChart

JavaFX - Bar Chart

A bar chart is used to represent grouped data using rectangular bars. The length of these bars depicts the values. The bars in the bar chart can be plotted vertically or horizontally.

Following is a bar chart, comparing various car brands.



In JavaFX, a Bar chart is represented by a class named **BarChart**. This class belongs to the package **javafx.scene.chart**. By instantiating this class, you can create an BarChart node in JavaFX.

Example

The following example depicts various car statistics with the help of a bar chart. Following is a list of car brands along with their different characteristics, which we will show using a bar chart –

Car	Speed	User Rating	Millage	Safety
Fiat	1.0	3.0	5.0	5.0
Audi	5.0	6.0	10.0	4.0
Ford	4.0	2.0	3.0	6.0

Following is a Java program which generates a bar chart, depicting the above data using JavaFX.

Save this code in a file with the name **BarChartExample.java**.

```java
//BarChartExample

import java.util.Arrays;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.stage.Stage;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;

public class BarChartExample extends Application {
    @Override
    public void start (Stage stage) {
        //Defining the axes
        CategoryAxis xAxis = new CategoryAxis();
        xAxis.setCategories(FXCollections.<String>observableArrayList(
            Arrays.asList("Speed", "User rating", "Milage", "Safety")));
        xAxis.setLabel("category");

        NumberAxis yAxis = new NumberAxis();
        yAxis.setLabel("score");
```

Define the X and Y axis of the bar chart and set labels to them. In our example, X axis represents the category of comparison and the y axis represents the score.

The constructor of this class, pass the objects representing the X and Y axis.

Instantiate the **XYChart.Series** class and add the data (a series of, x and y coordinates) to the Observable list of this class.

```
26      //Creating the Bar chart
27      BarChart<String, Number> barChart = new BarChart<>(xAxis, yAxis);
28      barChart.setTitle("Comparison between various cars");
29
30      //Prepare XYChart.Series objects by setting data
31      XYChart.Series<String, Number> series1 = new XYChart.Series<>();
32      series1.setName("Fiat");
33      series1.getData().add(new XYChart.Data<>("Speed", 1.0));
34      series1.getData().add(new XYChart.Data<>("User rating", 3.0));
35      series1.getData().add(new XYChart.Data<>("Milage", 5.0));
36      series1.getData().add(new XYChart.Data<>("Safety", 5.0));
37
38      XYChart.Series<String, Number> series2 = new XYChart.Series<>();
39      series2.setName("Audi");
40      series2.getData().add(new XYChart.Data<>("Speed", 5.0));
41      series2.getData().add(new XYChart.Data<>("User rating", 6.0));
42      series2.getData().add(new XYChart.Data<>("Milage", 10.0));
43      series2.getData().add(new XYChart.Data<>("Safety", 4.0));
44
45      XYChart.Series<String, Number> series3 = new XYChart.Series<>();
46      series3.setName("Ford");
47      series3.getData().add(new XYChart.Data<>("Speed", 4.0));
48      series3.getData().add(new XYChart.Data<>("User rating", 2.0));
49      series3.getData().add(new XYChart.Data<>("Milage", 3.0));
50      series3.getData().add(new XYChart.Data<>("Safety", 6.0));
51
```

```
52          //Setting the data to bar chart
53          barChart.getData().addAll(series1, series2, series3);
54
55          //Creating a Group object
56          Group root = new Group(barChart);
57
58          //Creating a scene object
59          Scene scene = new Scene(root, 600, 400);
60
61          //Setting title to the Stage
62          stage.setTitle("Bar Chart");
63
64          //Adding scene to the stage
65          stage.setScene(scene);
66
67          //Displaying the contents of the stage
68          stage.show();
69      }
70
71      public static void main (String args[]) {
72          launch(args);
73      }
74  }
```

Add the data series prepared to the bar chart.

Create group object (extends **javafx.scene)** Pass the BarChart (node) object as a parameter to the constructor of the Group class.

Create a Scene with width and height

You can set the title to the stage using the **setTitle()** method of the **Stage** class.

You can add a Scene object to the stage using **setScene()** and then show the contents

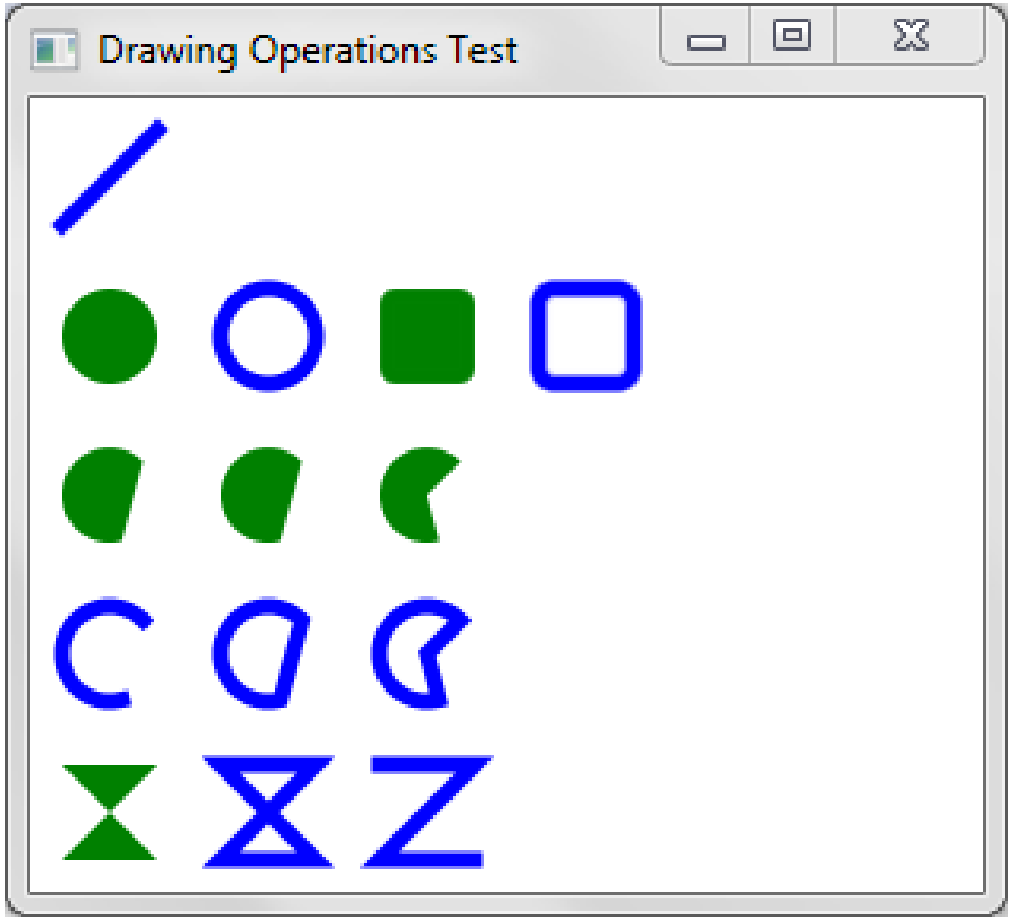
Finally, aunch the JavaFX application by calling the static method **launch()** of the **Application** class from the main method.

$ javac BarChartExample.java
$ java BarChartExample

Graphics

2D Graphics

Charts are 2D graphics (you also have 3D charts but if you ever read Tufte you'll never want to use them), but in charts you haven't full freedom to draw whatever you want on the screen. If you want to draw you should you a Canvas object. "Canvas" was the name of the cloth used in the old days for making ship sails. Put on a wooden frame, this is what western artists started to use around the 17th century for painting, hence the name in graphical interfaces.



Canvas Object

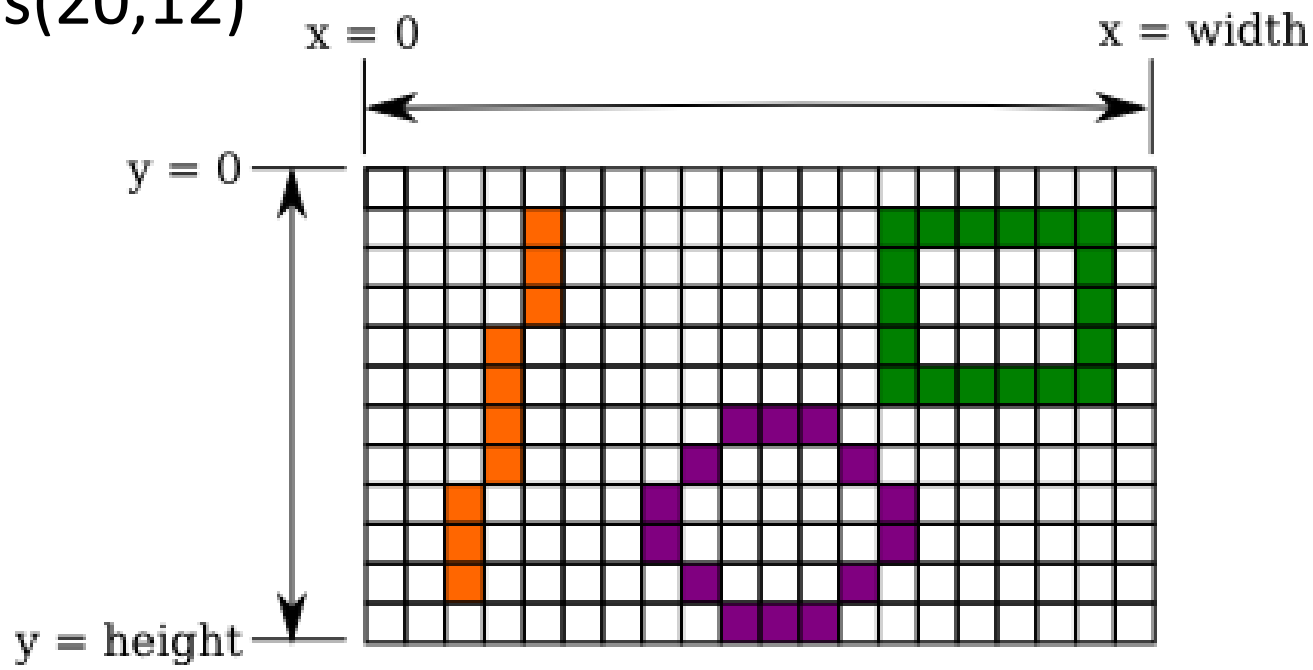
```
import javafx.scene.*;
import javafx.scene.paint.*;
import javafx.scene.canvas.*;
```

- Lines and Shapes

Canvas Object

The width and height of a *Canvas* can be specified in the constructor that used to create the canvas object. For example, to create a tiny 20-by-12 canvas:

Canvas canvas = new Canvas(20,12)



RandomStrings.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.scene.layout.BorderPane;
5  import javafx.scene.layout.StackPane;
6  import javafx.stage.Stage;
7  import javafx.scene.text.*;
8  import javafx.scene.canvas.*;
9  import javafx.scene.paint.Color;
10
11 /**
12  * This program displays 25 copies of a message.  The color and
13  * position of each message is selected at random.  The font
14  * of each message is randomly chosen from among five possible
15  * fonts.  The messages are displayed on a white background.
16  * There is a button that the user can click to redraw the
17  * image using new random values.
18  */
19 public class RandomStrings extends Application {
20
21     private final static String MESSAGE = "Hello JavaFX";
22
23     private Font font1, font2, font3, font4, font5;  // The five font
24
25     private Canvas canvas;  // The canvas on which the strings are dr
26
27
28     public static void main(String[] args) {
29         launch(args);
30     }
31
```

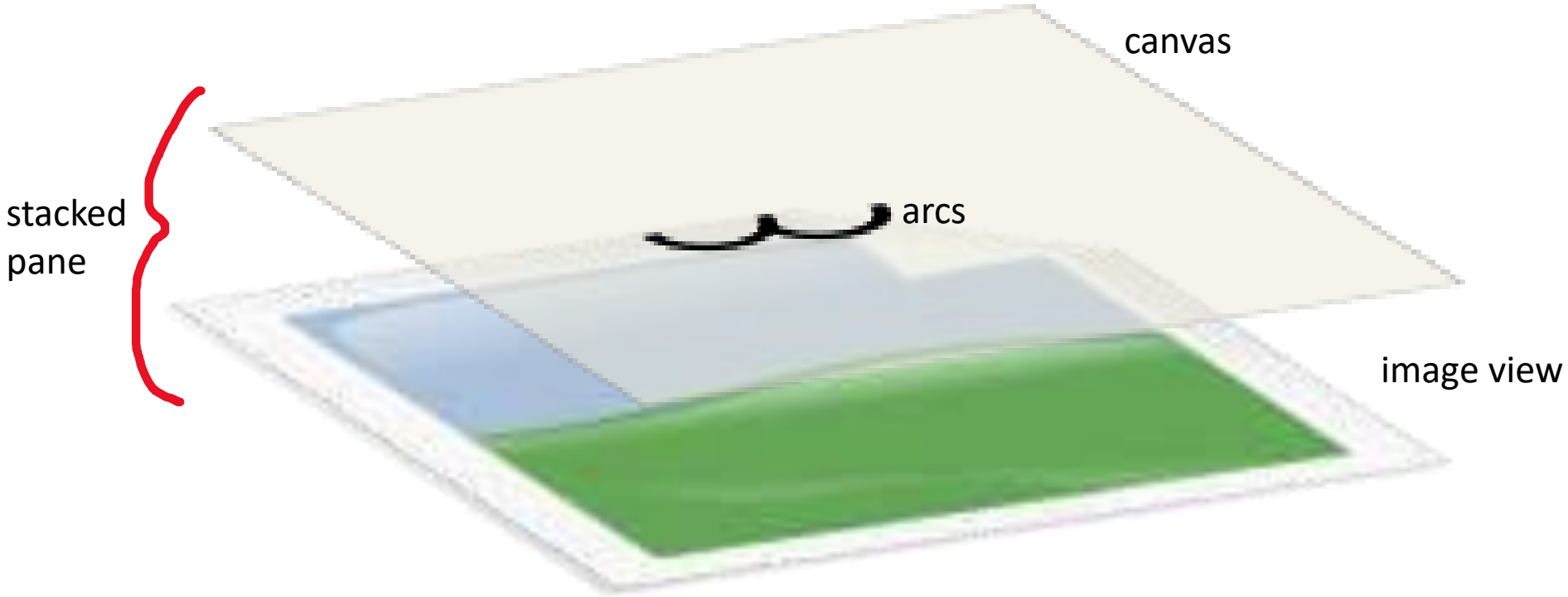
```
33      public void start( Stage stage ) {
34
35          font1 = Font.font("Times New Roman", FontWeight.BOLD, 20);
36          font2 = Font.font("Arial", FontWeight.BOLD, FontPosture.ITALIC, 28);
37          font3 = Font.font("Verdana", 32);
38          font4 = Font.font(40);
39          font5 = Font.font("Times New Roman", FontWeight.BOLD, FontPosture.ITALIC, 6
40
41          canvas = new Canvas(500,300);
42          draw();  // draw content of canvas the first time.
43
44          Button redraw = new Button("Redraw!");
45          redraw.setOnAction( e -> draw() );
46
47          StackPane bottom = new StackPane(redraw);
48          bottom.setStyle("-fx-background-color: gray; -fx-padding:5px;" +
49                          " -fx-border-color:black; -fx-border-width: 2px 0 0 0")
50          BorderPane root = new BorderPane(canvas);
51          root.setBottom(bottom);
52          root.setStyle("-fx-border-color:black; -fx-border-width: 2px");
53
54          stage.setScene( new Scene(root, Color.BLACK) );
55          stage.setTitle("Random Strings");
56          stage.setResizable(false);
57          stage.show();
58      }
59
```

```
60
61      /**
62       * The draw() method is responsible for drawing the content of the canvas.
63       * It draws 25 copies of the message string, using a random color, font, and
64       * position for each string.
65       */
66      private void draw() {
67
68          GraphicsContext g = canvas.getGraphicsContext2D();
69
70          double width = canvas.getWidth();
71          double height = canvas.getHeight();
72
73          g.setFill( Color.WHITE );  // fill with white background
74          g.fillRect(0, 0, width, height);
75
76          for (int i = 0; i < 25; i++) {
77
78              // Draw one string.  First, set the font to be one of the five
79              // available fonts, at random.
80
81              int fontNum = (int)(5*Math.random()) + 1;
82              switch (fontNum) {
83              case 1:
84                  g.setFont(font1);
85                  break;
86              case 2:
87                  g.setFont(font2);
88                  break;
89              case 3:
90                  g.setFont(font3);
91                  break;
92              case 4:
93                  g.setFont(font4);
94                  break;
95              case 5:
96                  g.setFont(font5);
97                  break;
98              } // end switch
99
00              // Set the color to a bright, saturated color, with random hue.
01
02              double hue = 360*Math.random();
03              g.setFill( Color.hsb(hue, 1.0, 1.0) );
04
```

```
99
100             // Set the color to a bright, saturated color, with random hue.
101
102             double hue = 360*Math.random();
103             g.setFill( Color.hsb(hue, 1.0, 1.0) );
104
105             // Select the position of the string, at random.
106
107             double x,y;
108             x = -50 + Math.random()*(width+40);
109             y = Math.random()*(height+20);
110
111             // Draw the message.
112
113             g.fillText(MESSAGE,x,y);
114
115             // Also stroke the outline of the strings with black
116
117             g.setStroke(Color.BLACK);
118             g.strokeText(MESSAGE,x,y);
119
120         } // end for
121
122     } // end draw()
123
124
```

Another example:

Take a background image, and Canvas on which to draw two half circles close to each other (part-circles shapes are called "arc")



Import packages

```
1  import javafx.application.Application;
2  import javafx.event.ActionEvent;
3  import javafx.event.EventHandler;
4  import javafx.scene.*;
5  import javafx.scene.layout.*;
6  import javafx.scene.paint.*;
7  import javafx.scene.canvas.*;
8  import javafx.scene.shape.*;
9  import javafx.stage.Stage;
10 import javafx.stage.Screen;
11 import javafx.scene.image.*;
12 import java.net.URL;
13
14 public class StupidCanvasExample extends Application {
15
16     public static void main(String[] args) { launch(args);
17
18 }
```

Credit: Stephane Faroult

```
19
20  public void start(Stage stage) {
21      double width;
22      double height;
23      double x;
24      double y;
25
26      stage.setTitle("StupidCanvasExample");
27      stage.setResizable(false);
28      Group root = new Group();
29      Scene scene = new Scene(root);
30      StackPane pane = new StackPane();
31
32      URL url = this.getClass() .getClassLoader()
33                                .getResource("background.jpeg");
```

StackPane to put image and Canvas (transparent) on top of it.

```
33                          .getResource("background.jpeg");
34
35      if (url != null) {
36          Image image = new Image(url.toString());
37          width = image.getWidth();
38          height = image.getHeight();
39          ImageView iv = new ImageView(image); pane.getChildren().add(iv);
40          final Canvas canvas = new Canvas(width, height);
41          GraphicsContext gc = canvas.getGraphicsContext2D();
```

To draw on a Canvas, you need the associated "GraphicsContext". This is where you define, among other things, line thickness and colours.

```
40          final Canvas canvas = new Canvas(width, height);
41          GraphicsContext gc = canvas.getGraphicsContext2D();
42
43          gc.setStroke(Color.BLACK);
44          gc.setLineWidth(height * 0.01);
45          x = 0.42 * width - width / 36.0;
46          gc.strokeArc(x, y,
47                       width / 18.0, height / 40.0,
48                       180, 180, ArcType.OPEN);
49          pane.getChildren().add(canvas);
50          // Make canvas disappear when clicked
51          canvas.setOnMouseClicked((e)->{ canvas.setVisible(false);});
```

There are multiple ways to define colors, here we use an enum.

Make the canvas invisible when clicked

```
49          pane.getChildren().add(canvas);
50          // Make canvas disappear when clicked
51          canvas.setOnMouseClicked((e)->{ canvas.setVisible(false);});
52
53          }
54
55          root.getChildren().add(pane);
56          stage.setScene(scene); stage.show();
57      }
58  }
59
60
```

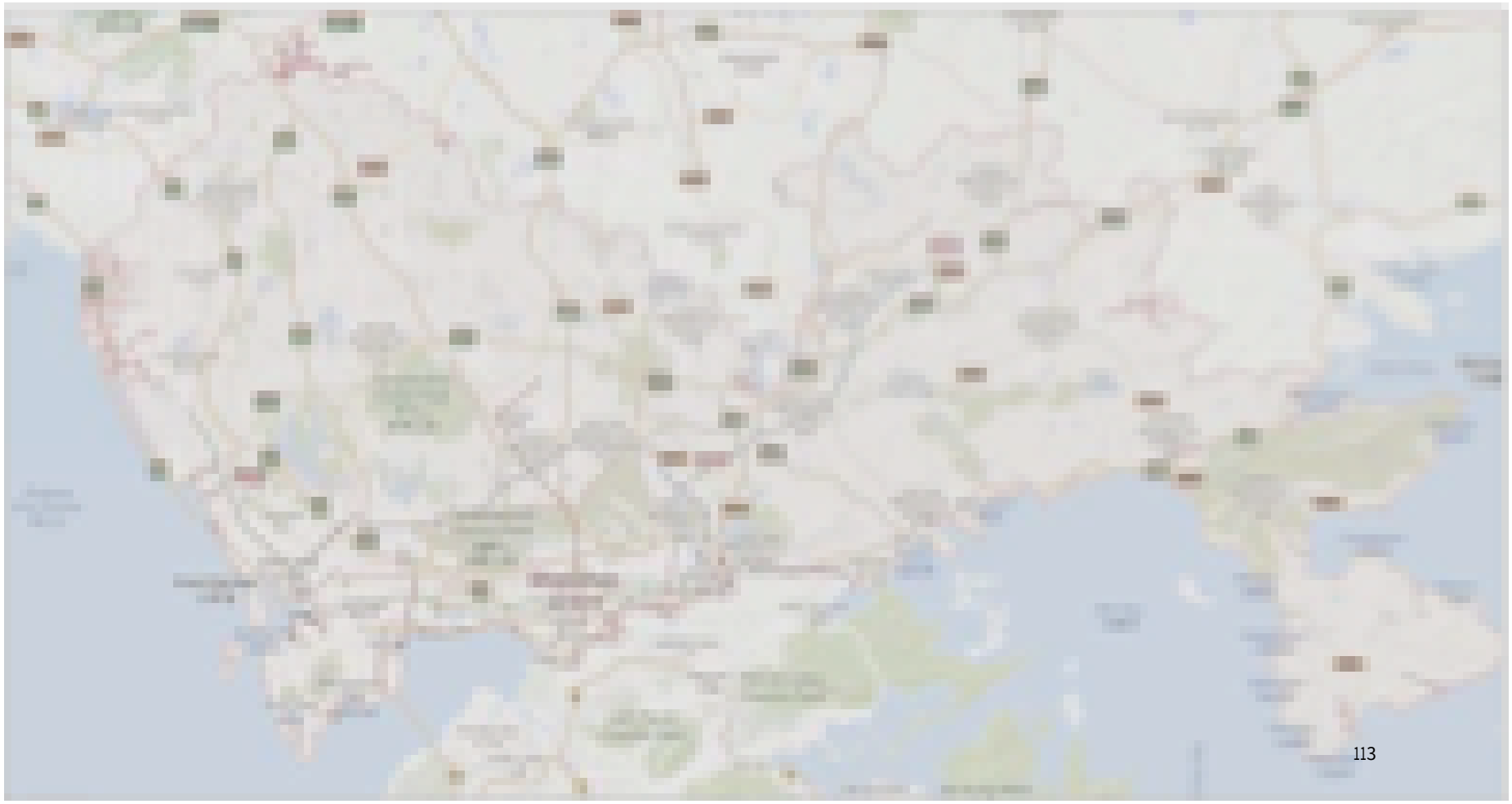
And there you go. All the art, of course, is in the choice of the suitable background image

If instead of using Mona Lisa you use a map, you can create some really interesting applications with canvases (other than a drawing tool).

Use a map as a background

Draw routes

You could have for instance one Canvas per Metro line, stack all of them, and use buttons to make a line appear or disappear. That said, working with maps is not very easy because what you want to plot are usually places for which you know latitude and longitude.



Interaction?

Shape Objects:

- Arc
  Circle
- Line
- Polygon
- Rectangle
- Text
- ...

You can only interact with a "Node". A Canvas is a node, and you can interact with it (I was able to make the Canvas over Mona Lisa invisible by clicking on it). However, you cannot directly interact with the shapes drawn over the canvas using graphics context. If you want to do this you need shape objects.

Using shape objects…

Stroke

Color .setStroke(col)

Width .setStrokeWidth(w)

+ other properties

You can set for them what you can set in the GraphicsContext of a Canvas (note that the Width of a Stroke is the line thickness)

Stroke

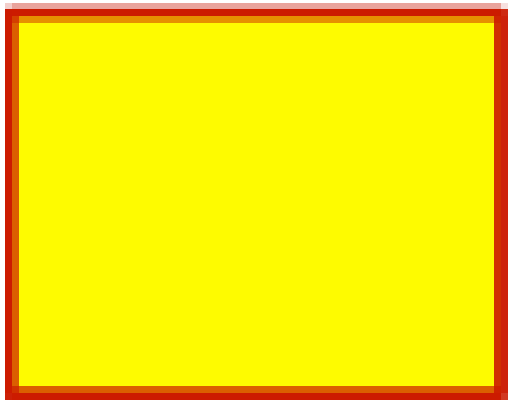
Color .setStroke(col)

Width .setStrokeWidth(w)

+ other properties

Fill:

color,
gradient,
image,
pattern



You can fill shapes too, colors can also be specified by an intensity of red, green, and blue, as well as a parameter that specifies transparency.

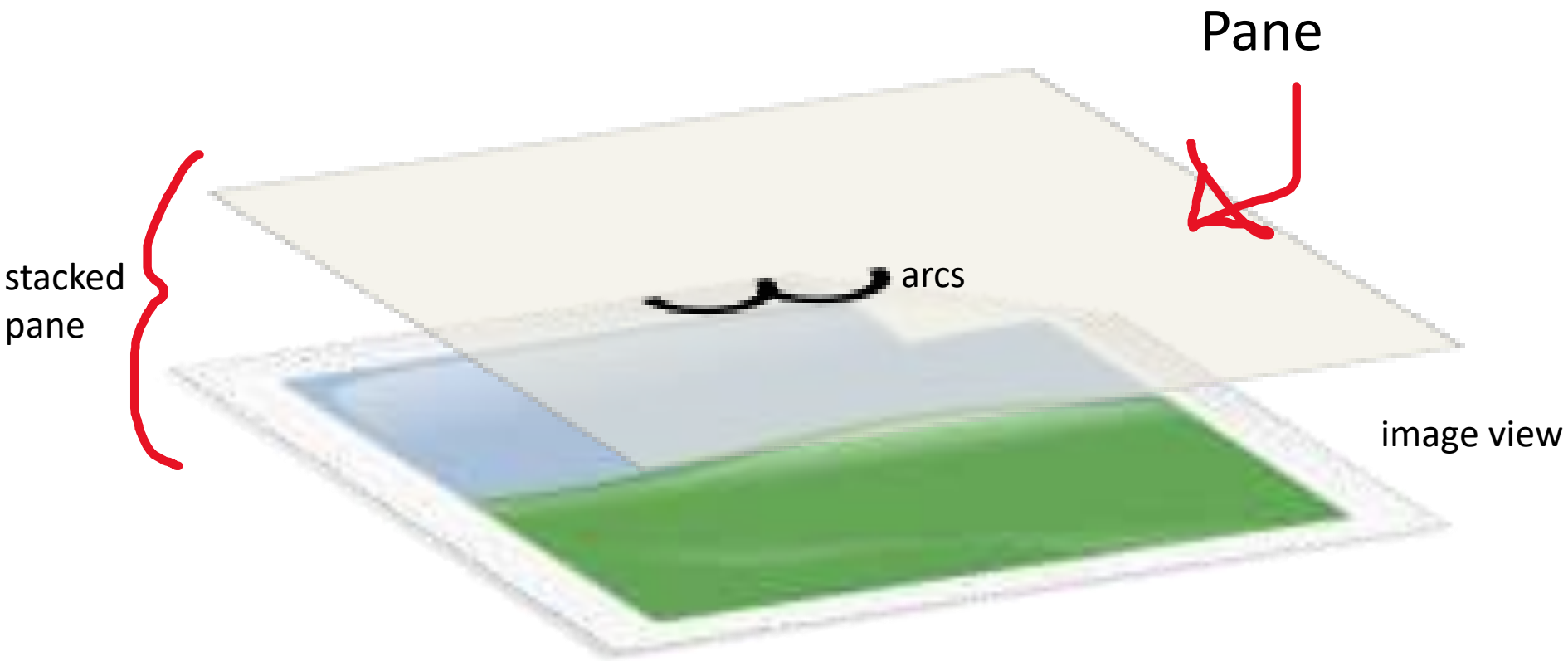
Color.rgb(255, 255, 0, 1.0)

Shapes are nodes ….

CLICKABLE

With shapes you can click on any individual shape.

We can redo the Monalisa with shapes instead of a Canvas…



```
43      Arc arc = new Arc();
44      arc.setCenterX(0.42 * width);
45      arc.setCenterY(0.288 * height);
46      arc.setRadiusX(width / 36.0);
47      arc.setRadiusY(width / 36.0);
48      arc.setStartAngle(180.0);
49      arc.setLength(180.0);
50      arc.setType(ArcType.OPEN);
51      arc.setStroke(Color.BLACK);
52      arc.setStrokeWidth(height * 0.01);
53      arc.setFill(Color.rgb(255, 255, 255, 0.0));
54      shapePane.getChildren().add(arc);
55
```

Same for arc2…

```
57  arc.setOnMouseClicked((e)->{
58      Random rand = new Random();
59      int r = rand.nextInt(256);
60      int g = rand.nextInt(256);
61      int b = rand.nextInt(256);
62      Color col = Color.rgb(r, g, b);
63      arc.setStroke(col);
64      arc2.setStroke(col);
65
```

Here the same action is associated with arc and arc2

Canvas or Shapes

- The choice depends how many shapes there are
- If too many shapes will become slow and difficult to check if you clicked in the right place…
- Note: its also possible to check where a canvas was clicked:

mouseEvent .getX() / .getY()

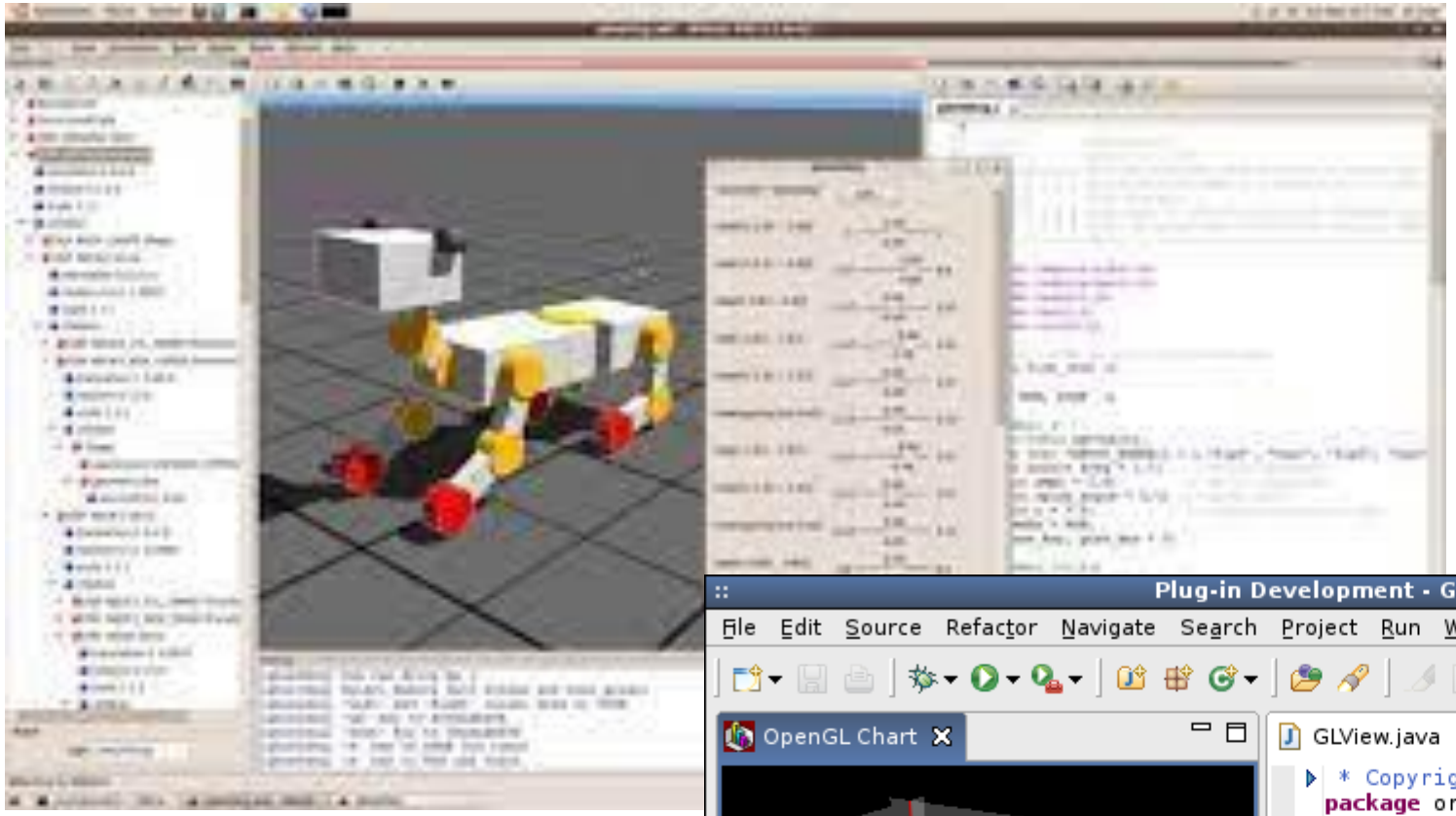
3D Graphics

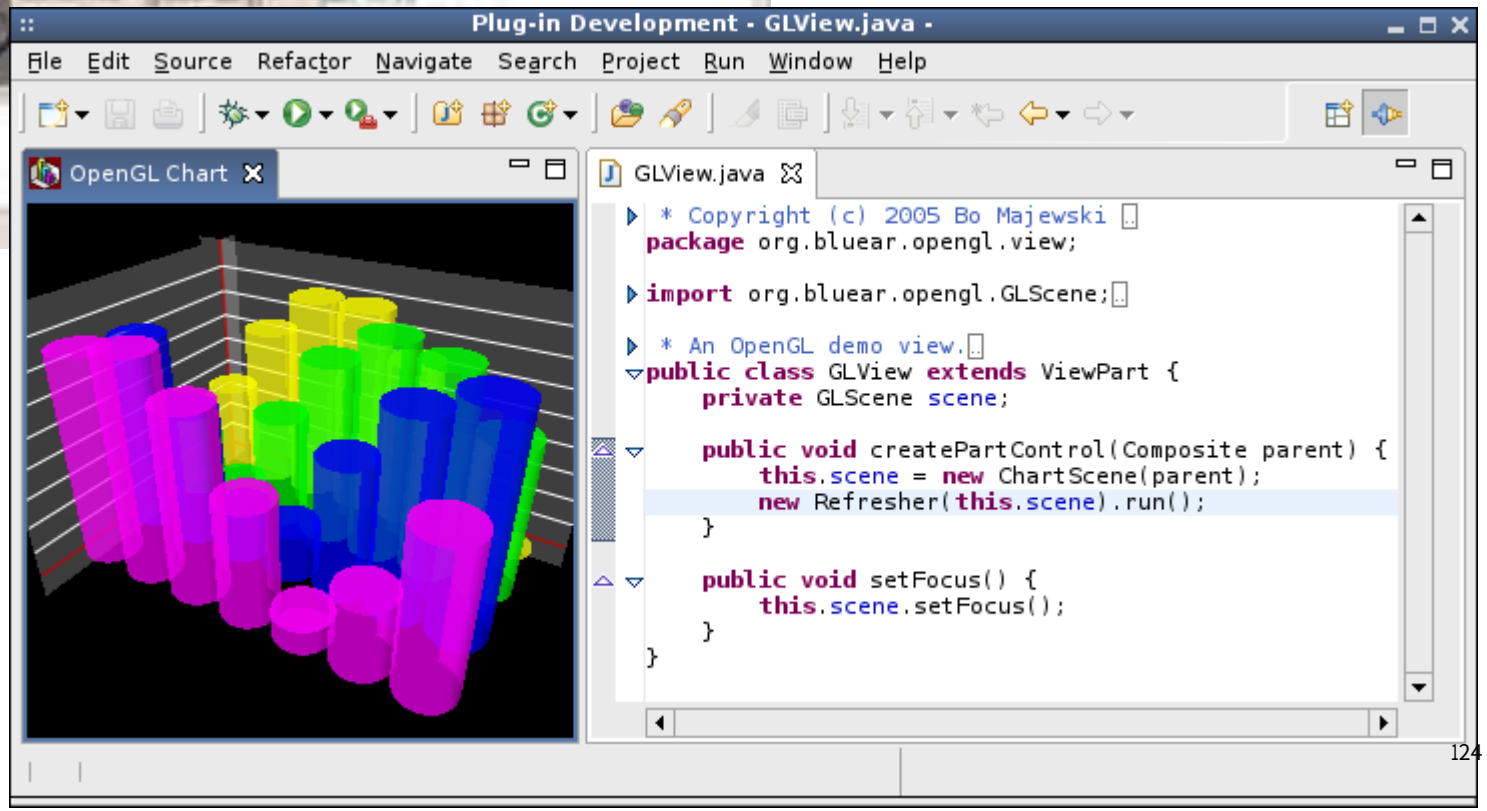
- JavaFX can also render 3D graphics

e.g. javafx.scene.shape.Shape3D

We won’t discuss 3D graphics further because it's beginning to become very specific to advanced applications but there are packages in java which make rendering 3D graphics relatively straight forward for specific applications.


SK LINE




Plug-in Development - GLView.java -
File Edit Source Refactor Navigate Search Project Run Window Help
OpenGL Chart
GLView.java
* Copyright (c) 2005 Bo Majewski
package org.bluear.opengl.view;
import org.bluear.opengl.GLScene;
* An OpenGL demo view.
public class GLView extends ViewPart {
private GLScene scene;
public void createPartControl(Composite parent) {
this.scene = new ChartScene(parent);
new Refresher(this.scene).run();
}
public void setFocus() {
this.scene.setFocus();
}
}

Multimedia

Media

You can also play audio and video in JavaFX. It's not very different from images. With images you have an Image object, and the ImageView that shows it on screen. With audio and video, you have a Media object, you have a MediaView, and between the two you have a MediaPlayer object with controls allowing you to start, pause, stop, rewind and so forth.

Very much like Images

- Media
- MediaPlayer  Controls
- MediaView

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.geometry.Pos;
import javafx.util.Duration;

import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
```

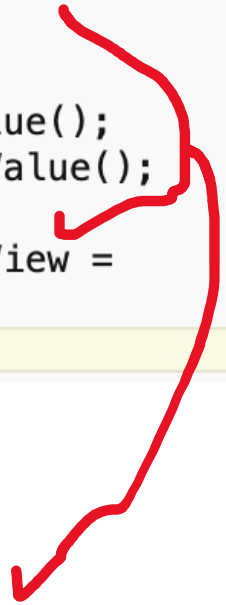
```
14  public class MediaDemo extends Application {
15
16  private final String MEDIA_URL = this.getClass()
17                                       .getClassLoader()
18                                       .getResource("TestVid.mp4")
19                                       .toString();
20
21
22  private final String MEDIA_URL = "http://edu.konagora.com/video/TestVid.mp4";
```

URL, Path and String

- URL: prefix://path
- PATH: path

A Media (like an Image) can take a URL as argument of a constructor. An URL (like an URI, basically the same thing) is a prefix + a path. If the prefix is "file:" it means that the resource (... name given to anything you can load) is accessible through the file system of your computer (it's not necessarily local, it can be a network disk). It can also be something else such as "http:" to mean that the resource is accessed from a web server through HTTP requests. You usually need to apply toString() to them.

```
21
22  private final String MEDIA_URL = "http://edu.konagora.com/video/TestVid.mp4";
23
24
25  @Override
26  public void start(Stage primaryStage) {
27      Media media = new Media(MEDIA_URL);
28      int width = media.widthProperty().intValue();
29      int height = media.heightProperty().intValue();
30      MediaPlayer mediaPlayer =
31      new MediaPlayer(media); MediaView mediaView =
32      new MediaView(mediaPlayer);
33      Button playButton = new Button(">");
```



Once the media is loaded, you associate it with a MediaPlayer, and the MediaPlayer with a MediaView. Controls will execute MediaPlayer methods.

```
new MediaView(mediaPlayer);
Button playButton = new Button(">");

playButton.setOnAction(e -> {
if (playButton.getText().equals(">")) {
        mediaPlayer.play();
        playButton.setText("||"); }else{
        mediaPlayer.pause();
        playButton.setText(">"); }
        });

Button rewindButton = new Button("<<");
rewindButton.setOnAction(e->
        mediaPlayer.seek(Duration.ZERO));

Slider slVolume = new Slider();
```

The text on the button tells us what is the current state, and whether we should play or pause.
I'm also adding another button for rewinding, and a new widget (Slider) for setting the volume.

```
54
55      HBox hBox = new HBox(10);
56      hBox.setAlignment(Pos.CENTER); hBox.getChildren().addAll(playButton,
57                  rewindButton,
58      new Label("Volume"), slVolume);
59      BorderPane pane = new BorderPane();
60
61      pane.setCenter(mediaView);
62      pane.setBottom(hBox);
63      Scene scene = new Scene(pane, 750, 500);
64      primaryStage.setTitle("MediaDemo");
65      primaryStage.setScene(scene);
66      primaryStage.show();
67      }
68
69
70      public  void main(String[] args) {
71          launch(args);
72      }
73  }
74
75
```

Other than geometry (size) I give the range and initial value for the slider (0 to 100, initially 50) and "bind" it to the MediaPlayer. There is an implicit ChangeListener behind, to change the volume when the slider moves.

Controls are in a box, everything is added to a BorderPane (that controls placement as top/right/bottom/left and center) and we are ready to go.

This demo and the slides are by Stephane Faroult


MediaView
MediaDemo
MediaPlayer
The Media is added to a MediaPlayer that provides methods for controlling the media (playing it, pausing it and so forth). The MediaPlayer is in turn added to a MediaView which is what JavaFx can display.
Media
I have in my demo application added to a "border Pane" the MediaView and a Hbox (Horizontal box) that contains widgets that call the MediaPlayer methods.
Hbox for controls

Style Sheets

Skins

- Last JavaFx subject, superficial in all meanings of the word but important (looks sell): how to change the appearance of a JavaFx application. I have already briefly talked about it, the best way is to do it through an external style sheet (.css file). People will be able to change, often in a very impressive way, the looks of your application by changing this file and without any need to access the code (in fact, they just need the .css and the .class to run the "modified" application).
- If you want to see how far you go with "styling", you can visit http://csszengarden.com and click on designs on the right hand- side. The same page will look completely different.

CSS ZEN GARDEN

The Beauty of CSS Design

Select a Design:

Mid Century Modern by Andrew Lohman

Garments by Dan Mall

Steel by Steffen Knoeller

Apothecary by Trent Walton

Screen Filler by Elliot Jay Stocks

Fountain Kiss by Jeremy Carlson

A Robot Named Jimmy by meltmedia

Cascading Style Sheet (CSS)

CSS stands for Cascading Style Sheet and Cascade is French for waterfall.
The name emphasizes that styles "drip down”, and are inherited from previous style sheets

It all starts with HTML…

The idea is stolen from web applications. Web applications just send HTML pages to browsers that decode and display these pages. An HTML page is organized by sections between pairs of tags (<tag> at the beginning, </tag> at the end) that structure the document and can contain in turn other pairs of tags, thus defining a kind of hierarchical structure (note that some tags, such as those for images, act both as opening and closing tags). Tags pretty often also contain attributes (such as the image file name for an image tag).

<tags>

In the very early days of the web, people were using tags to format their pages, for instance what they wanted in bold was between **<bold>** and **</bold>** (inspired by previous document generation systems that were sending special signals to printers), and you could change the fonts with **<font** *attributes specifying the font, size and everything***>** ... ****. As websites were growing in size and number of pages, and as increasingly pages were generated by programs instead of being created by hand, it became unmanageable, especially when the marketing department was deciding on new corporate colors. So the idea was to associate formatting to tags in one or several separate text files.

Some people could just focus on the "engine", such as here an application allowing to retrieve film information from a database ...

Film Database

Title: Mermaid

Director

Actor/Actress

Country

Year

And others can change the looks by just changing a .css file

Submit Query

Title	Country	Year	Directed By	Starring	Also Known As
Million Dollar Mermaid	United States	1952	Mervyn LeRoy	Victor Mature,Walter Pidgeon,Esther Williams	
La Sirène du Mississipi	France	1969		Jean Paul Belmondo,Catherine Deneuve	Mississippi Mermaid
Ningyo Densetsu	Japan	1984	Toshiharu Ikeda	Junko Miyashita,Mari Shiraki	Mermaid Legend,人魚伝
The Little Mermaid	United States	1989	Ron Clements,John Musker	Christopher Daniel Barnes,Jodi Benson,Pat Carroll,Paddi Edwards,Buddy Hackett,Jodie MacClung,Kenneth Mars,Samuel E. Wright	
Mei ren yu	China	2016	Stephen Chow	Choo Deng,Yuxi Lin,Shaw Luo,Yuqi Zhang	The Mermaid,美人鱼

The way it works in web pages

- Before I discuss about CSS in JavaFx, I'm going to talk about CSS with HTML, because there is far more CSS written for HTML than for JavaFX.
- There are three main ways to specify how to display what is between tags.
- 1. You can specify for a given tag, associating a tagname with visual characteristics
  2. I have mentioned that tags can have attributes, one is "class" (unrelated to object-oriented programming) listing one or several categories. This allows for creating a subcategory, or to give some common visual characteristics across different tags that have the same class (for instance "inactive")
- 3. You can give another attribute "id" to a tag, an this allows you to make one particular tag look really special.

<tag>

```
tag {
    attribute: value;
    ...
}
```

I focus on simple tags in the next example.

<tag class="category">

You have here how the tag looks in HTML and how styling looks in CSS.

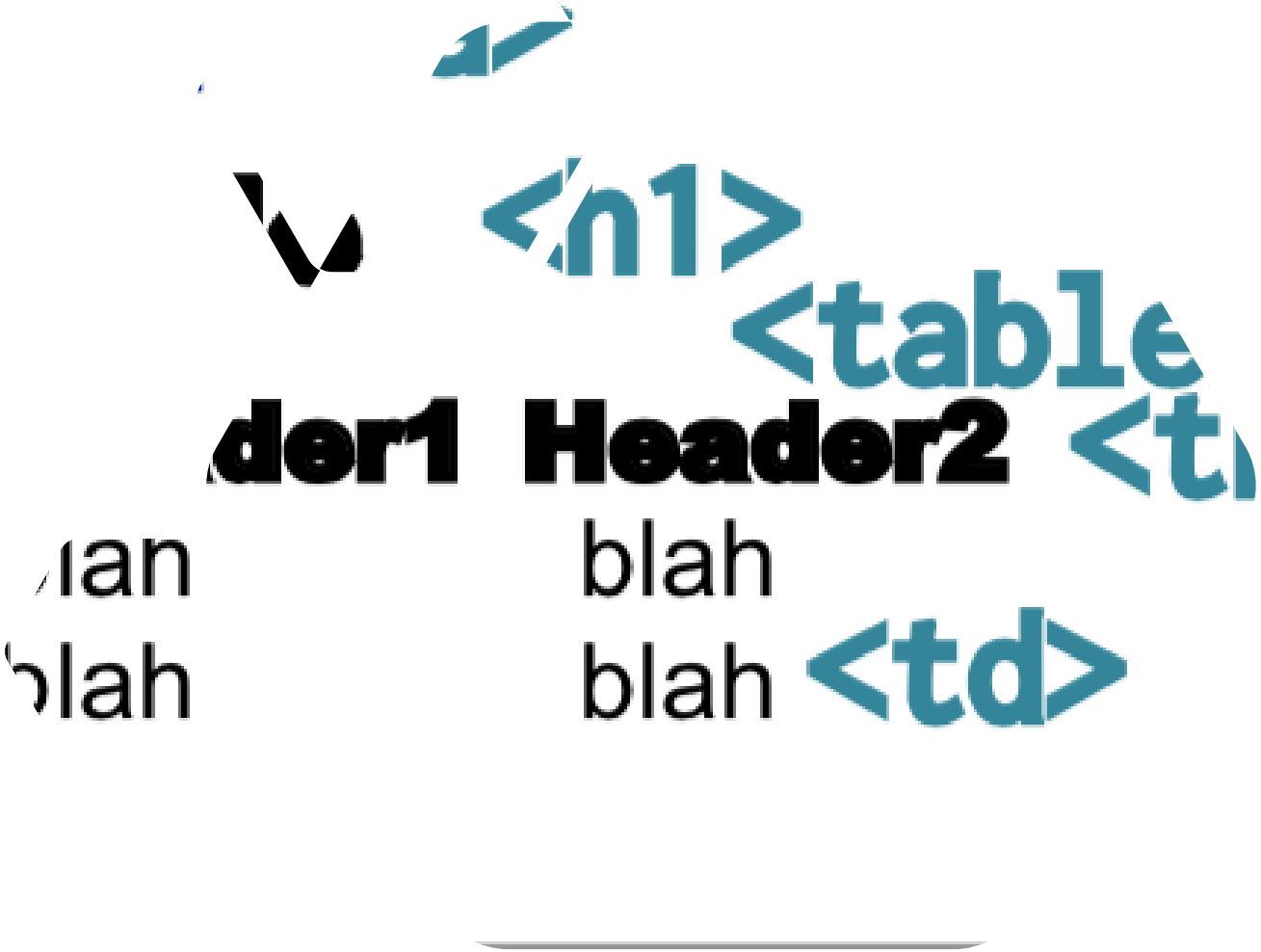
```
.category {
    attribute: value;
    ...
}
```

<tag id="name">

```
#name {
    attribute: value;
    ...
}
```

<body>

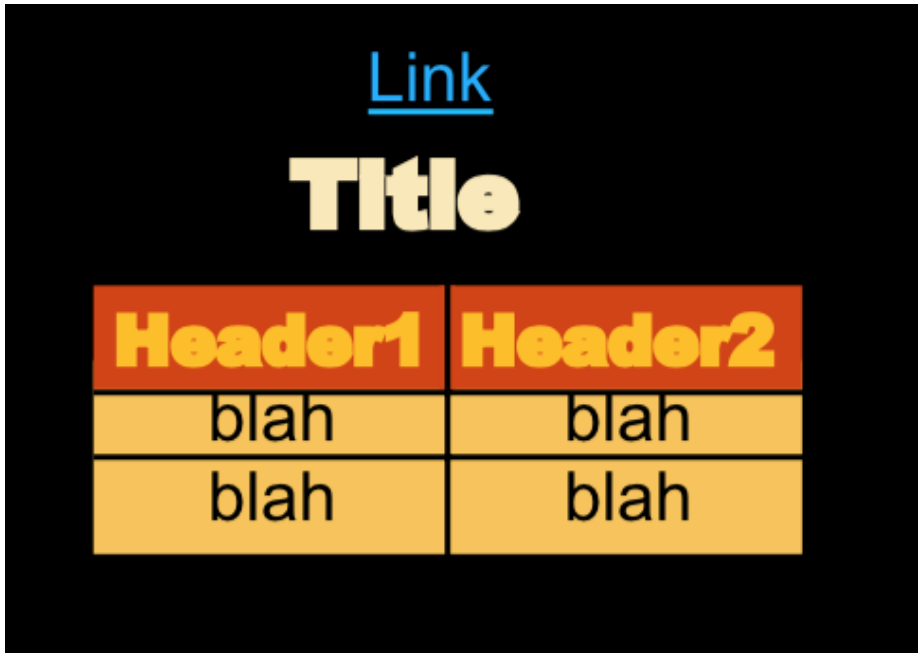
- In a HTML page, everything that is displayed is between <body> and </body> tags, so <body> really defines the global looks of the page.
- In my example, I'll use tags <a> associated with a link, <h1> with a (first level) header, <table>, <th> (table header) and <td> (table data). I have omitted <tr> (table row) which usually surrounds the columns of a same row. The tag will be there on the page, but I'm not associating any special style with it in my example.



Notice the special a:visited (link you have already clicked on) and a:hover (when the cursor moves over the link)

```
body {text-align: center;
        background-color: black;}
a {color: lightskyblue;}
a:visited {color: lightseagreen;}
a:hover {color: salmon;}
h1 {color: cornsilk;}
th {background-color: sienna;
    color: orange;}
td {background-color: burlywood;}
table {width: 80%;
    margin-left: auto;
    margin-right: auto;
    text-align: center;}
form {width: 50%;
        margin-left: auto;
        margin-right: auto;
        padding: 20px;
        background-color: silver;}
```

styles.css



<link rel="stylesheet" href="styles.css"/>

Styling is written to a file that is included in the HTML page by a special tag in the <header>...</header> section that precedes the "body". Then you can change it when needed.

The way it works in JavaFX

Nodes are (almost) equivalent to tags

- .root plays the same role as body
- Otherwise use class names prefixed with a dot
- .button

You have of course no tags in a JavaFx application, but you have the same kind of hierarchy through nodes, containers and widgets. The JavaFx class names are used with the same syntax as the HTML classes in CSS, that means that they are prefixed by a dot.

The way it works in JavaFX

Nodes are (almost) equivalent to tags

.root plays the same role as body

Otherwise use class names prefixed with a dot

.button

Attribute names are prefixed with –fx-

-fx-font-size: 150%;

CSS attributes also have a special name with JavaFx. The idea is to be able to have a single CSS files shared by a Web and a JavaFx application without having any conflict.

The way it works in JavaFX

Nodes are (almost) equivalent to tags

.root plays the same role as body

Otherwise use class names prefixed with a dot

.button

Attribute names are prefixed with –fx-

-fx-font-size: 150%;

Node.setId("name")

Node.getStyleClass().add("css class")

Finally, node methods allow you to associate with a node the same kind of attributes as with a HTML tag. You can only have a single Id, but you can have several classes, and therefore getSyleClass() returns a list.

The way it works in JavaFX

Nodes are (almost) equivalent to tags

.root plays the same role as body

Otherwise use class names prefixed with a dot

.button

Attribute names are prefixed with –fx-

-fx-font-size: 150%;

Node.setId("name")

Node.getStyleClass().add("css class")

Node.setStyle("-fx-attribute: value")

```
Scene scene = new Scene(new Group(), 500, 400);
scene.getStylesheets().add("path/styles.css");
```

(then load into the application)

Not Everything cannot be styled with CSS in JavaFX

CSS styling allows you to go rather far in JavaFx, but not as far as you could go in HTML. Some elements may prove hard to style with CSS. You may sometimes have to code some styling in the Java application. However, if you still want this styling to be "externalized", don't forget that properties files also provide a way to read attributes at run-time. It's of course better to have all styling at one place, but it's better to use a properties file than to hard-code.

A Properties file might sometimes be a workaround