CISC 3120

# C12: JavaFX Scene Graph, Events, and UI Components

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# Outline

- Recap and issues
- JavaFX build-in UI elements
  - Simple event registration and handler
- Assignments

## Recap and Issues

- Projects
  - Project 1 & 2
  - Upcoming project: project 3
    - GUI application
- Midterm Review
  - Review guides and take-home tests
- GUI and Overview of JavaFX

# Lessons from Project 1

- Java naming convention
  - How should you name constants and variables?
  - How objects should interact with each other?
- Reduce maintenance difficulty
  - Using literals
    - Named constants are better
  - Divide-and-conquer: writing methods and classes
- Bottom-up and top-down approaches
  - When unclear, write few, run/test often
  - Consider how each part interacts with each other

# Naming Constants and Variables

- Which one of the two should you write according to the Java coding convention?

```
final static int GAME_BOARD_WIDTH = 80;
```

```
final static int gameBoardWidth = 80;
```

- Which one of the two should you write?

```
int GAME_BOARD_WIDTH = 80;
```

```
int gameBoardWidth = 80;
```

## Naming Constants

- Which one of the two should you write according to the Java coding convention?

  - `final static int GAME_BOARD_WIDTH = 80;` ✓
  - `final static int gameBoardWidth = 80;` ✗

- Which one of the two should you write?

  - `int GAME_BOARD_WIDTH = 80;` ✗
  - `int gameBoardWidth = 80;` ✓

## Using Literals

- Which one is easier to understand when you read?

```
if (numGuesses < 10) {
    …
}
```

```
final static int MAX_ALLOWED_GUESSES = 10;

…

if (numGuesses < MAX_ALLOWED_GUESSES) {
…
}
```

# Divide-and-Conquer: Writing Methods

- Which one is easier to read and code?

```
public class TreasureHuntGameConsoleApp
{
    public static void main(String[] args)    {
        CommandLineParser parser = new DefaultParser();
        int gameWidth = 80, gameHeight = 25, gameLevel = 0;
        Options options=new Options();
        options.addOption("w","width", true,"width parameter");
        options.addOption("h","height", true,"height parameter");
        options.addOption("l","level", true,"level parameter");

       try {
           CommandLine line = parser.parse(options, args);
           if(!(line.getOptionValue("w")==null))
               w = line.getOptionValue("w");
               gameWidth = Integer.parseInt(w);
               ……
       } catch (ParseException exp) {
       }
      GameController controller  =
          new
              GameController(gameWidth,gameHeight,gameLevel);
          controller.runTheGame();
    }
}
```

```
public class TreasureHuntGameConsoleApp
{

    public static void main(String[] args)    {
       parseGameOptions(args);
      GameController controller  =
        new
           GameController(gameWidth,gameHeight,gameLevel);
        controller.runTheGame();    }


    private static void parseGameOptions(String[] args) { …



    ……



   }

   private static int gameWidth;
   private static int gameHeight;
   private static int gameLevel;
}
```

# Questions?

- Lessons from Project 1
  - Java naming convention, using constants, method invocation, divide-and-conquer, and bottom-up/top-down

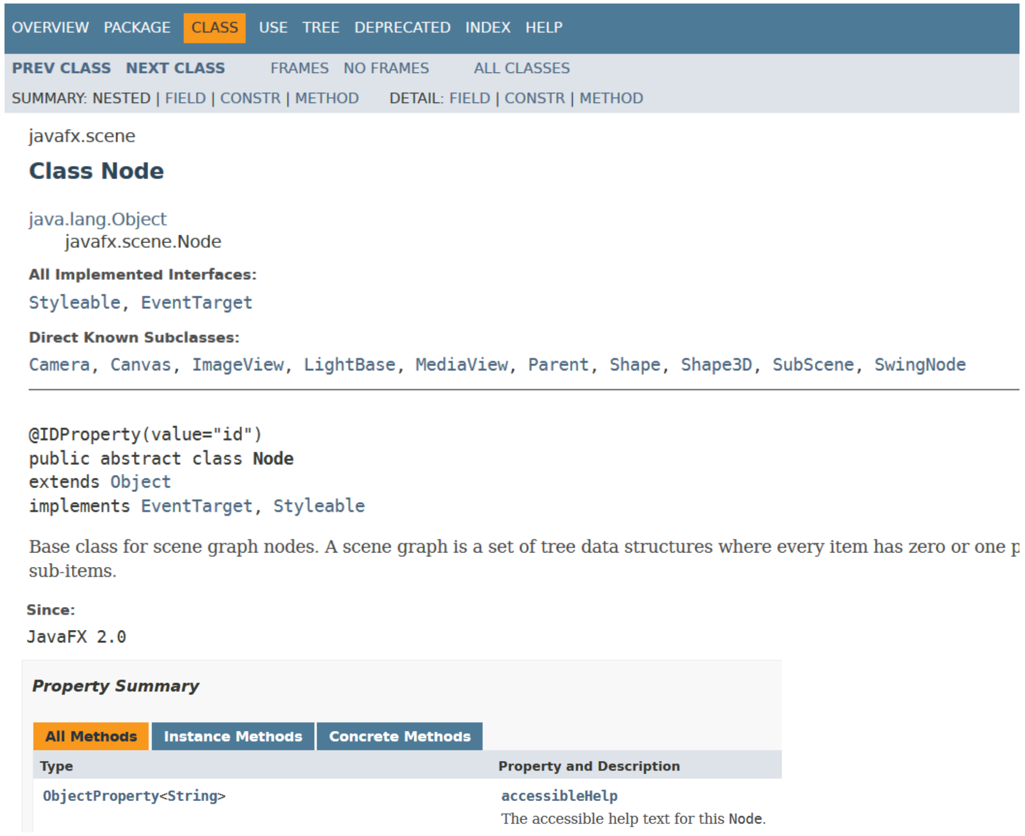
"Programs must be written for people to read, and only incidentally for machines to execute."

-- H. Abelson and G. Sussman (in "The Structure and Interpretation of Computer Programs")

# Java API Documentation

- Class documentation
  - Package hierarchy
  - Class name
  - Implemented interfaces
  - Known subclasses
  - Class declaration line
    - Abstract or concrete
    - Super class
  - Description
  - Compatibility
- Properties
  - Public instance variables
- Fields
  - Public class variables and constants
- Constructors
- Methods
  - Method summary
  - Methods inherited
- Property detail

# Java API Documentation



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

javafx.scene

**Class Node**

java.lang.Object
    javafx.scene.Node

**All Implemented Interfaces:**

`Styleable, EventTarget`

**Direct Known Subclasses:**

`Camera, Canvas, ImageView, LightBase, MediaView, Parent, Shape, Shape3D, SubScene, SwingNode`

```
@IDProperty(value="id")
public abstract class Node
extends Object
implements EventTarget, Styleable
```

Base class for scene graph nodes. A scene graph is a set of tree data structures where every item has zero or one p
sub-items.

**Since:**

`JavaFX 2.0`

***Property Summary***

| Type | Property and Description |
|---|---|
| `ObjectProperty<String>` | `accessibleHelp` The accessible help text for this `Node`. |

## Questions?

- How to consult API documentation?

# JavaFX GUI Application

- Learn to write JavaFX application
  - Learn new ones from existing knowledge and skills
  - Learn to use Java API documentation
  - Learn a few concepts in GUI and computer graphics
- JavaFX application life cycle
- JavaFX application structure
- JavaFX event processing
- JavaFX build-in UI components

## JavaFX Application

- JavaFX platform is the environment where JavaFX applications run
  - javafx.application.Platform: Application platform support class
    - Control & query platforms: e.g., accessibility, implicit exit
- Entry point: the Application class
  - javafx.application.Application
    - abstract void start(Stage primaryStage)

# JavaFX Application Life-Cycle

- JavaFX runtime does the following, in order,
  - Constructs an object of the specified Application class (via the launch(String[] args) method), with regard to the Application object:
  - Calls the *init()* method that can be overridden
  - Calls the *start(javafx.stage.Stage)* method that must be overridden in subclass)
  - Waits for the application to finish, which happens when either of the following occur:
    - the application calls *Platform.exit()*
    - the last window has been closed and the implicitExit attribute on Platform is true
  - Calls the stop() method (can be overridden)

# JavaFX Application: Remarks

- The *start(javafx.stage.Stage)* is an abstract method, and must be overridden in the subclass
- The init() and stop() method have concrete implementations, but do nothing, and can be overridden.
- Explicitly terminating JavaFX application
  - calling Platform.exit() is the preferred method
  - Calling System.exit(int) is acceptable, but the *stop()* method will ***not*** run.
- JavaFX should not and cannot be used after System.ext(int) is called or the stop() is returned.

## Questions?

- JavaFX Platform and Application
- Main agenda when developing JavaFX applications

# Stage and Scene

"All the world's a stage, and all the men and women merely players."

-- As You Like It, Act II, Scene VII, William Shakespeare

# JavaFX Stage

- Top level JavaFX container
    - Can have a Scene
    - Associated with a Window
- Primary Stage
    - First Stage constructed by the Platform
- Additional Stage
    - Constructed by the application

## Stage Style

- A stage can be one of a few styles
  - StageStyle.DECORATED
  - StageStyle.UNDECORATED
  - StageStyle.TRANSPARENT
    StageStyle.UTILITY

## Stage Modality

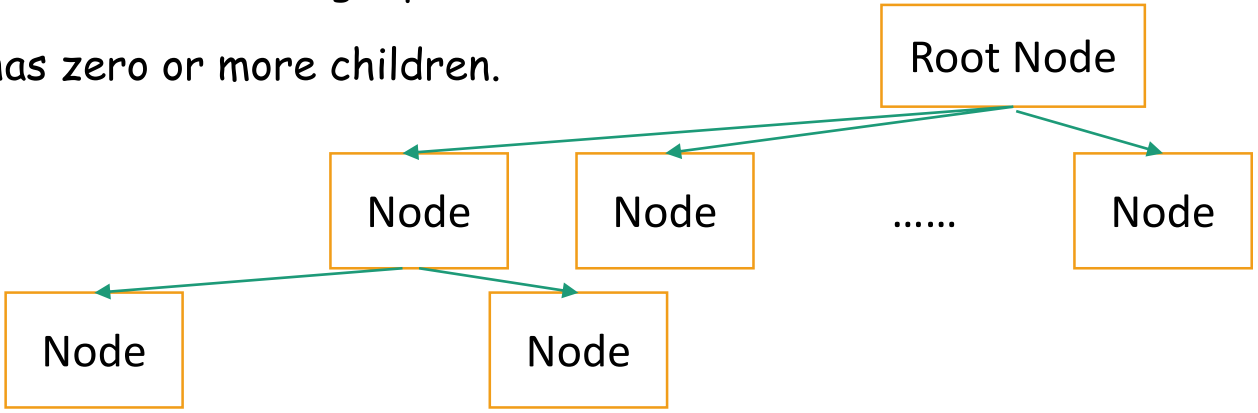
- Modality.NONE
- Modality.WINDOW_MODAL
- Modality.APPLICATION_MODAL

# JavaFX Scene

- Represent visual elements of user interface.
  - Elements can be displayed inside a window (on a Stage)
  - Scene graph
    - The elements form a graph called a scene graph
- Handles input via its elements
- Can be rendered.

# Scene Graph

- Elements organized as a hierarchical structure, like a tree (a tree is a graph)
  - A graph is understood as a collection of nodes (or vertices), and edges (representing some connection or association)
  - An element in a scene graph is called a node.
    - Each non-root node has a single parent.
    - Each node has zero or more children.



## Node in Scene Graph

- Example nodes
  - a layout container, a group, a shape, a button …
- Each node has an ID, style class, bounding volume, and other attributes
  - Effects, such as blurs and shadows
  - Opacity
  - Transforms
  - Event handlers (such as mouse, key and input method)
  - An application-specific state
- javafx.scene.Node: abstract class

## Building Scene Graph

- Create a root Node
- Add children Nodes to root Node
- Register event handlers
- Set it on a Stage

## Write the First JavaFX Application from Scratch

- Create a concrete subclass extending the JavaFX Application class (javafx.application.Application)
- (Curtains down) Construct a scene graph containing a tree of nodes
  - The simplest tree contains a single root node (select a concrete subclass of nodes)
    - http://docs.oracle.com/javase/8/javafx/api/javafx/scene/Node.html
  - Register some events to handle
- Set scene for the stage
- (Curtains up) Show the scene

## Questions

- JavaFX Stage
- JavaFX Scene
- Simple JavaFX application

## Building Scene Graph

- Packaged in javafx.scene
- Nodes (elements)
  - Examples: text, charts, containers, shapes (2-D and 3-D), images, media, embedded web browser, and groups
- Transforms
  - e.g., positioning and orientation of nodes
- Effects
  - Visual effects (algorithm resulting in an image)
  - Objects that change the appearance of scene graph nodes, such as blurs, shadows, and color adjustment
- A scene graph must have a root node

# Scene Graph Root Node

- Must a concrete subclass of javafx.scene.Parent
- Can be a Group or a Region object
  - Group
    - effects and transforms to be applied to a collection of child nodes.
  - Region
    - class for nodes that can be styled with CSS and layout children.
    - Layouts and Controls

# Layouts and Controls

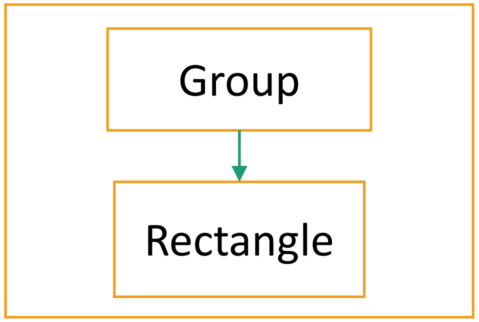
- Layouts
  - Classes support user interface layout
    - Examples: horizontal layout, vertical layout, grid layout, back-to-front
- Controls
  - A node in the scene graph that can be manipulated by the user
    - Labeled: buttons, labels, text fields, toggle button, checkbox, menu button, …
    - List view, combo box, menu bar, scroll bar, progress indicator, spinner, slider, …

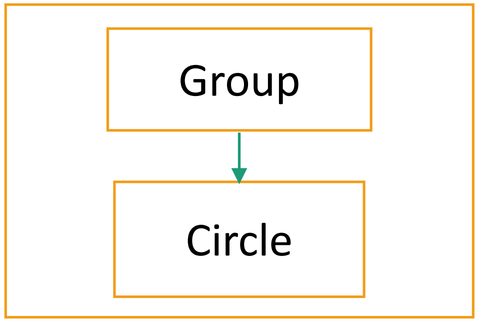
## Questions?

- Stage and Scene
- Scene graph
- GUI windows and Scene node

## Building a JavaFX Application with Stage and Scene: Example

- Can we have multiple scenes?
- How do we improve readability?
    - Use named constants
- Can we add more children to a scene graph?
- Can we have multiple stages (windows)?





## Questions?

- JavaFX scene graph
- Procedure to build a scene graph
- Review the example JavaFX applications
- Suggestion
  - Consult API documentation often

## Events

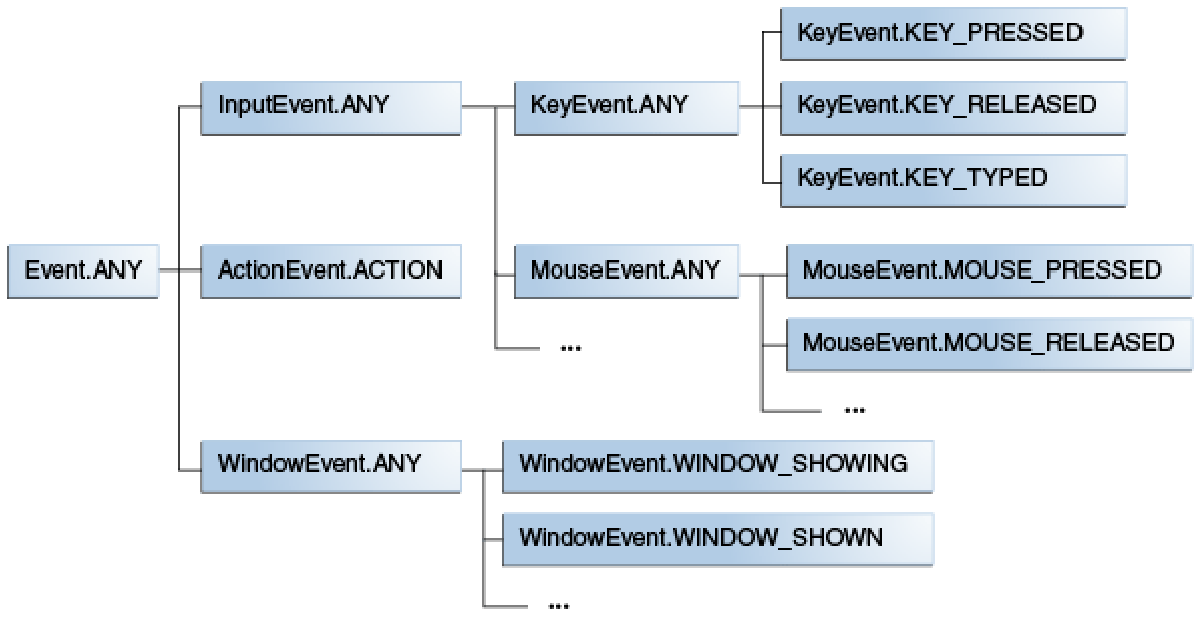
- Representing occurrence of something of the application’s interest
- Mouse events
  - Mouse pressed, mouse released, mouse clicked (pressed & released), mouse moved, mouse entered, mouse exited, mouse dragged
- Keyboard event
  - Key pressed, key released, key typed (pressed & released)
- Gesture event, touch event, …

## JavaFX Events

- javafx.event.Event
  - An event is an object of the Event class or any subclass of the Event class
- An event travels along a path called an event dispatcher chain
  - Typically, the path consists of objects of various Nodes, Stage, and Scene
- There are many types of events

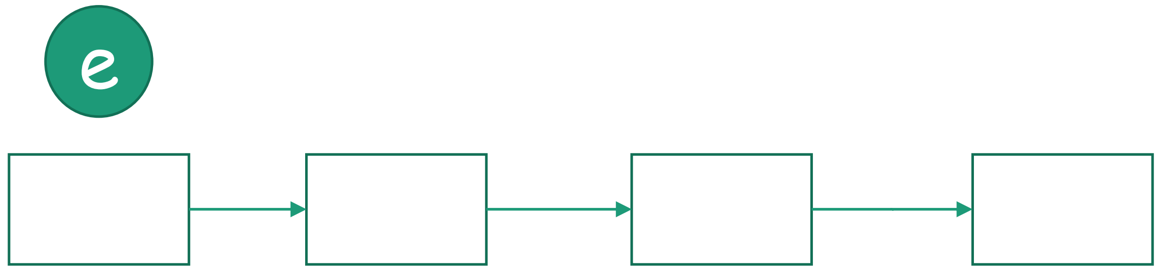
# JavaFX Event Type

- javafx.event.EventType: specific event type associated with an Event
  - Event types forms a hierarchy



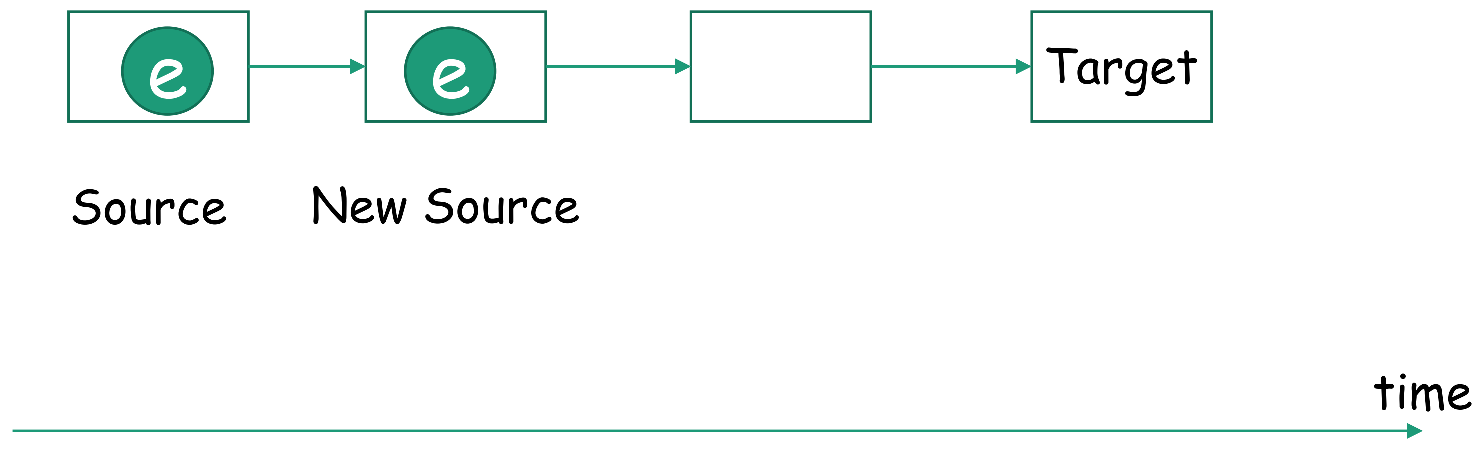
## Event Dispatcher Chain

- A path of nodes along which an event object is passed
- Event source
  - where (an object) an event is originated. The source changes as the event is passed along the chain
- Event target
  - a node where the action occurred and the end node of the dispatcher chain. The target does not change.



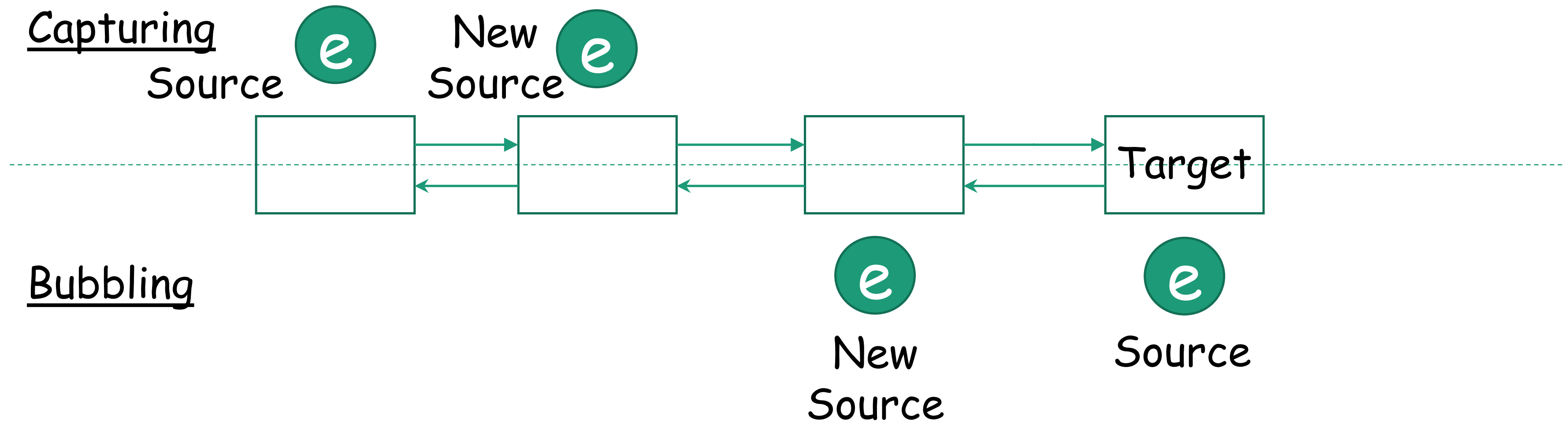
# Passing Event

- Passing an Event object along the dispatcher chain
  - The source changes as it travels
  - The target remains the same



# Event Capturing and Bubbling

- Undisturbed, an event is passed/travels in a two-way "round trip"
  - Capturing: source to target
  - Bubbling: target to source

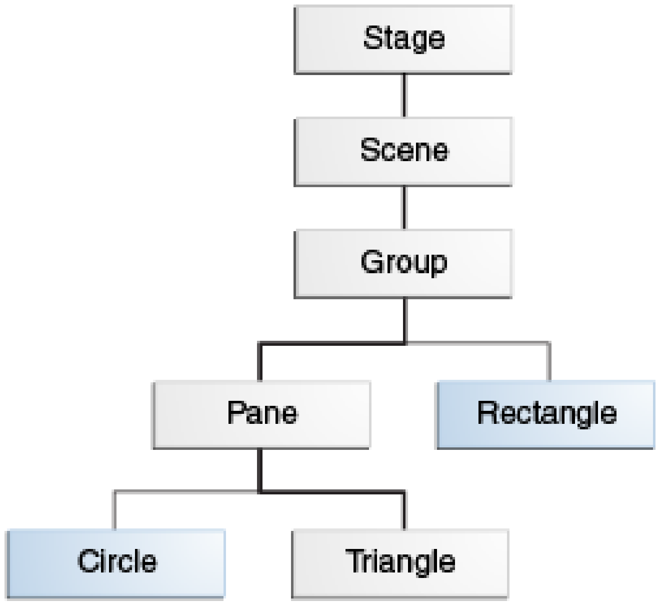


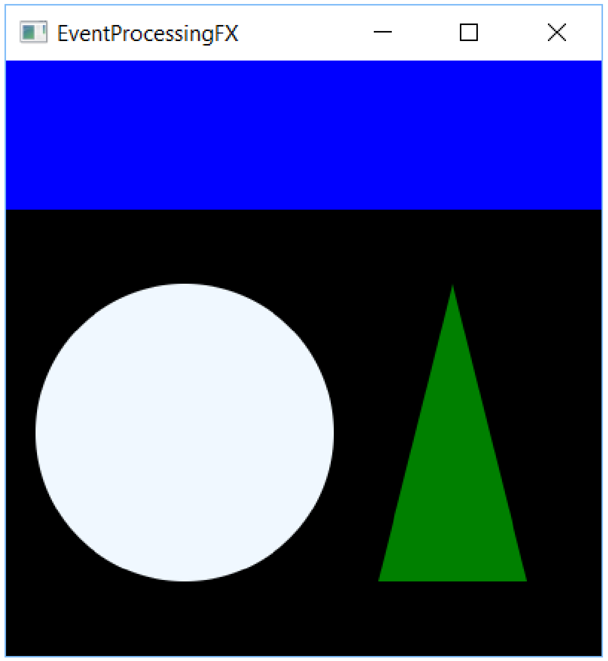
## Event Handling

- Event handling via EventFilter and EventHandler
- Add one or more EventFilter at each node
  - Invoked during the capturing phase
- Add one or more EventHandler at each node
  - Invoked during the bubbling phase

## Event Capturing and Bubbling: Example

- An implementation of the example in the Oracle's JavaFX tutorial





## Event Delivery Process

- Target selection
- Route construction
- Event capturing
- Event bubbling

## Target Selection

- When an action occurs, JavaFX determines which node is the target based on internal rules
- Examples:
    - Key events: the target is the node that has focus.
    - Mouse events: the node at the location of the cursor.
    - Gesture events: the node at the center point of all touches at the beginning of the gesture; or the node at the location of the cursor.
        - Swipe events: the node at the center of the entire path of all of the fingers; or or the node at the location of the cursor.
    - Touch events: the node at the location first pressed.
    - If more than one node is located at the cursor or touch, the topmost node is considered the target.

## Route Construction

- Selected event target determine the initial dispatch chain
    - It implements the buildEventDisptachChain(…) method in the EventTarget interface
    - The implementation of the method determines the initial chain

# Consume Events

- Events can be consumed
    - Stop passing the event to next node along the event dispatcher chain in either direction
        - event capturing
        - event bubbling
    - In an event filter
        - Stops capturing
    - In an event handler
        - Stops bubbling

## Question?

- JavaFX event handling
- Event dispatcher chain
  - Event source, event target, event capturing phase, event bubbling phase
- Default/initial event dispatcher chain constructed by Nodes
- Event handling
  - event handlers and event filters
- Events can be consumed

## Basic Event Handling

- Register event handlers at nodes
- Response to
  - mouse events, keyboard events, action events, drag-and-drop events, window events, and others.
- Commonly via the convenience methods provided by the target nodes
  - setOnMouseClicked, setOnMouseEntered, seOnMouseExited …
  - Naming convention: setOn<u>Event-type</u>

## Basic Event Handling: Examples

- Example programs in the Sample Programs repository

## Questions

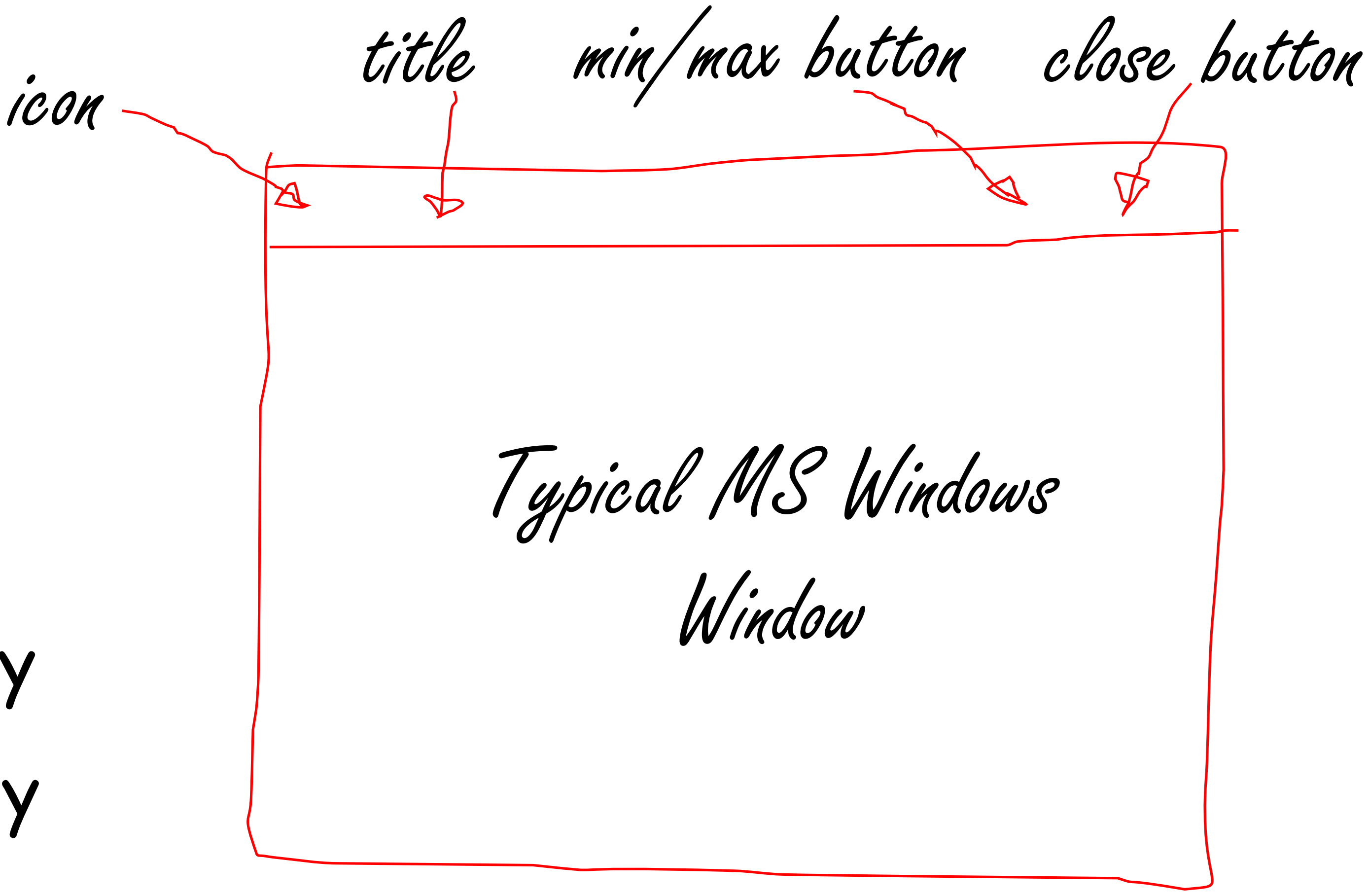
- Basic event handling
- Examples

# Write Larger JavaFX Applications

- Now, ready to engage in writing slightly larger applications in JavaFX
- A few essential concepts
  - Window & node coordinates, colors
- Use JavaFX build-in user interface components
- Design user interface and example application

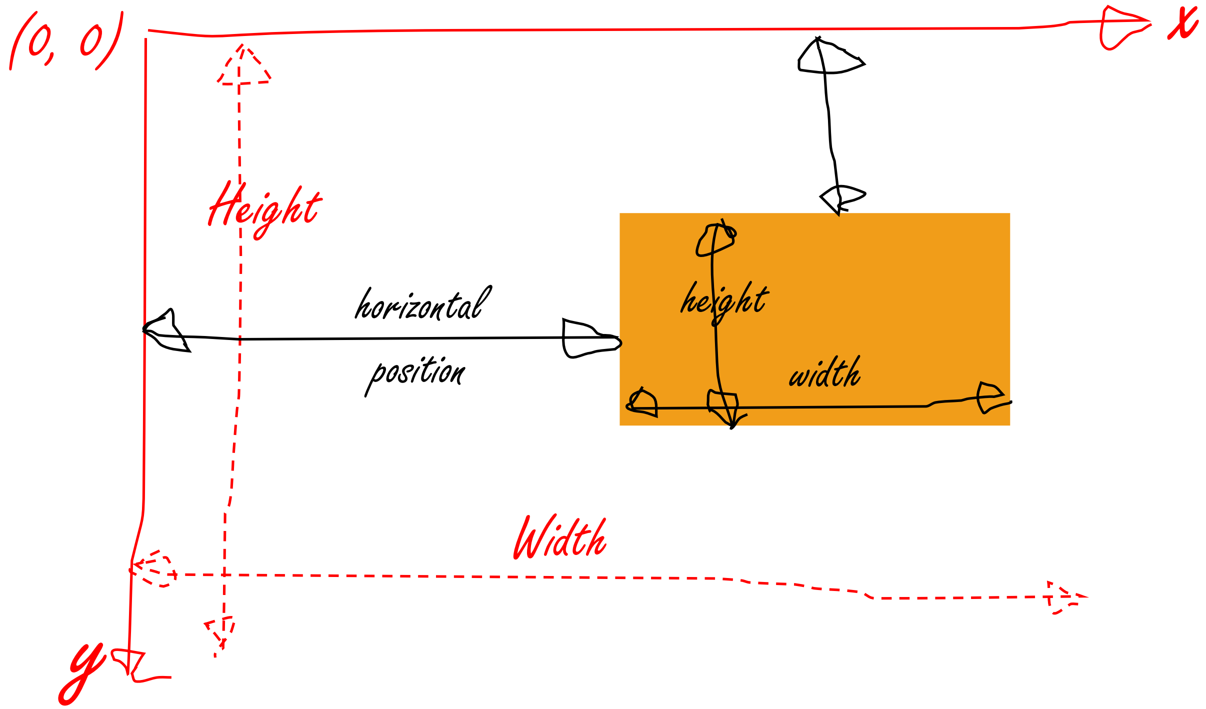
# GUI Windows

- Size
- Shape
- Title
- Icon
- Modality
- Visibility



## Scene Node Coordinate System

- A traditional computer graphics "local" coordinate system (javafx.scene.node)

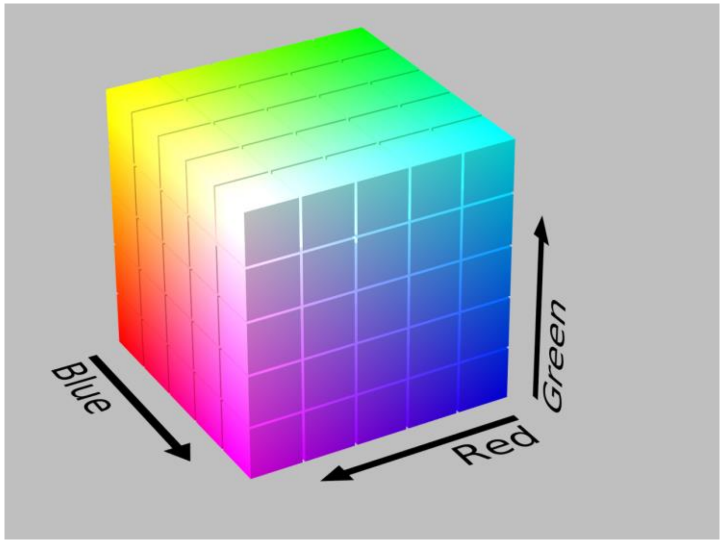


## Color Space

- Color is a human perception
- (Mathematical) models for color are developed
    - Including a model for human perceptual color space
    - Examples
        - Machine first
            - Additive: Red-Green-Blue (RGB)
            - Subtractive: Cyan-Magenta-Yellow-Black/Key (CMYK)
        - Human first
            - Hue-Saturation-Brightness (HSB)
        - Processing first
            - LAB (Luminance and a & b color channels)

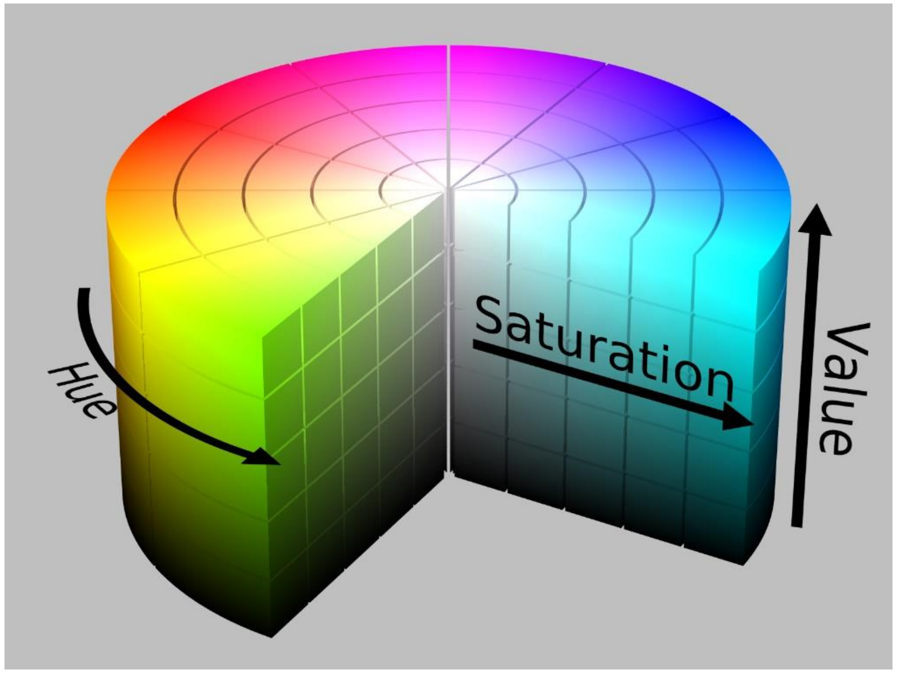
## Standard Red-Green-Blue (sRGB)

- Red, Green, Blue
    - 0. – 1.
- Alpha (transparency or opacity)
    - 0.0 – 1.0 or 0 – 255; 1. or 255
    - 0. or 0: completely opaque
    - 1. or 1: completely transparent



# Hue-Saturation-Brightness (HSB)

- Hue:
  - 0. – 360.
- Saturation:
  - 0. – 1.
- Brightness (or Value):
  - 0. – 1.
- Alpha (transparency or opacity)
  - 0.0 – 1.0 or 0 – 255; 1. or 255
  - 0. or 0: completely opaque
  - 1. or 1: completely transparent



## Color and Static Factory Method

- *A static method that returns an instance of the class*
  - Examples:
    - static Color hsb(double hue, double saturation, double brightness, double opacity)
    - static Color rgb(int red, int green, int blue, double opacity)
- In your application design: advantage and disadvantage?

## Blocking and Non-Blocking

- The show() method of a Stage object does not block the caller and returns “immediately”.
- The showAndWait() method of a Stage object shows the stage and waits for it to be hidden (closed) before returning to the caller.
  - Cannot be called on the primary stage

# Questions?

- Window coordinate system
- Blocking and non-blocking behaviors
- Color and color spaces

# User Interface Components

- Layouts
- UI controls
- Text
- Canvas and Shapes
- Images
- Charts
- HTML content & embedded web browser
- Groups

## Use Build-in UI Controls and Layouts

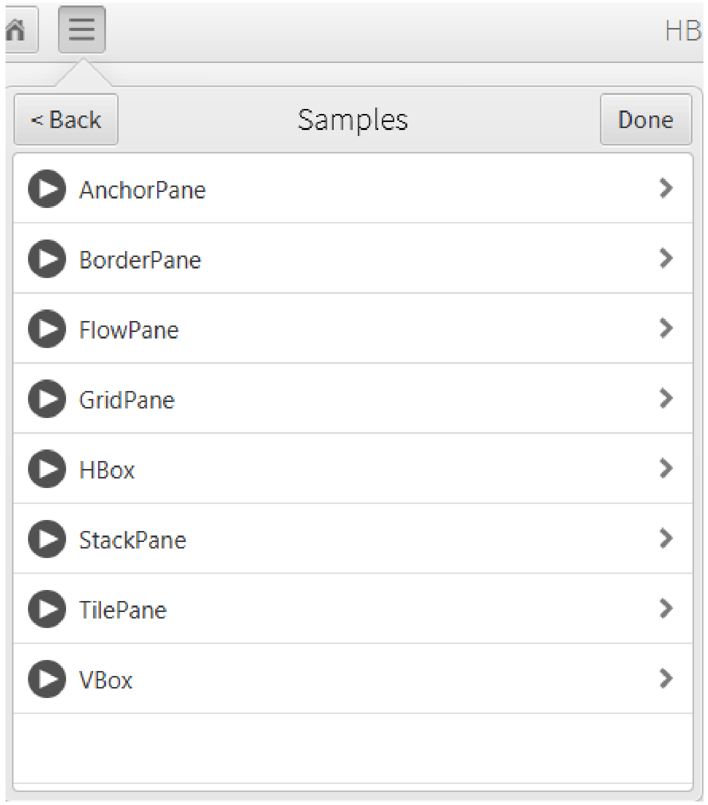
- Layout containers: prebuilt layouts for UI controls and more
- UI controls: prebuilt user interface controls
- Use texts
- Use 2D graphics
- Handle user interactions with simple event handlers

## Layout Containers (Panes)

- Packaged in javafx.scene.layout
- Arrangements of the UI controls within a scene graph
- Provide the following common layout models
  - BorderPane
  - HBox
  - VBox
  - StackPane
  - GridPane
  - FlowPane
  - TilePane
  - AnchorPane

# Explore Layouts

- Using the JavaFX Ensemble 8 sample application
  - Run the ensemble.EnsembleApp class

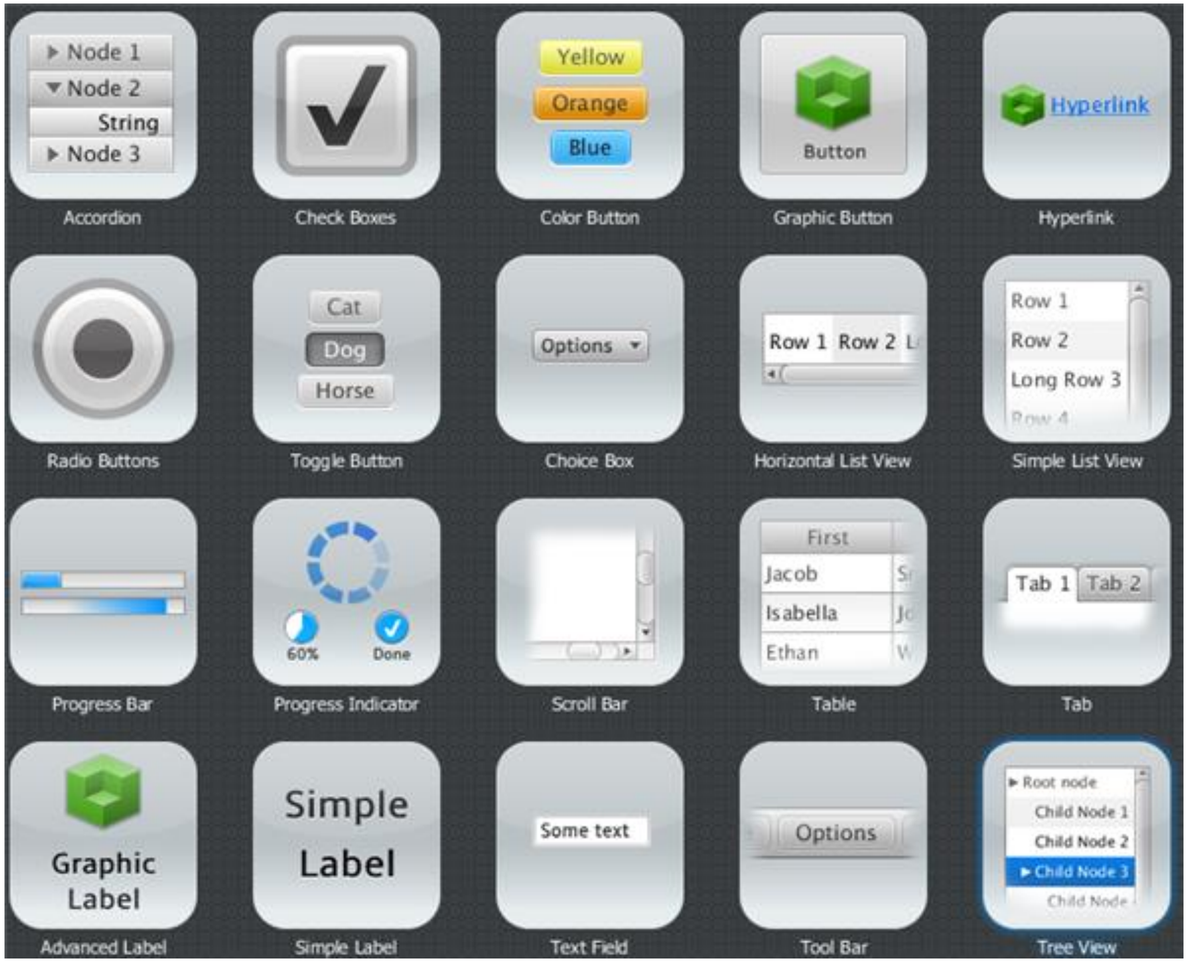


## UI Controls

- Packaged in javafx.scene.control

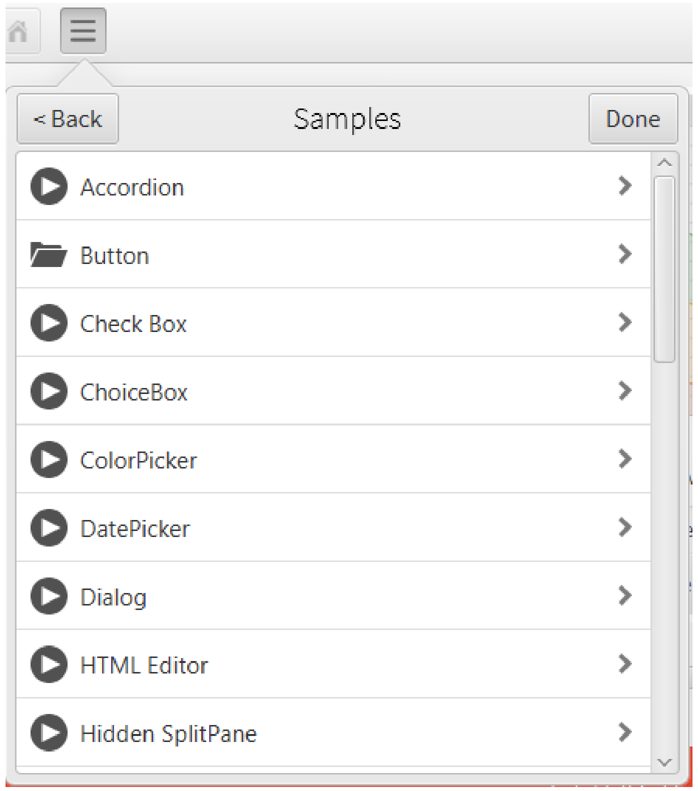
- Label
- Button
- Radio Button
- Toggle Button
- Checkbox
- Choice Box
- Text Field
- Password Field
- Scroll Bar
- Scroll Pane
- List View
- Table View
- Tree View
- Tree Table View
- ComboBox
- Separator
- Slider
- Progress Bar
- Progress Indicator
- Hyperlink
- Tooltip
- HTML Editor
- Titled Pane
- Accordion
- Menu
- Color Picker
- Date Picker
- Pagination Control
- File Chooser

# A Gallery of Selected UI Controls



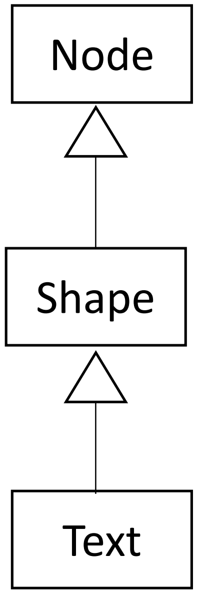
# Explore UI Controls

- Using the JavaFX Ensemble 8 sample application
  - Ensemble 8 is in the "Sample Programs" repository
    - Open it as a Maven project
  - Run the ensemble.EnsembleApp class



# Text

- Packaged in javafx.scene.text.Text
  - Text class inherits from the Shape class, and the Shape class inherits from the Node class
    - You can apply effects, animation, and transformations to text nodes in the same way as to any other Nodes.
    - you can set a stroke or apply a fill setting to text nodes in the same way as to any other Shapes.



# 2-D Graphics

- Draw images on Canvas
  - Canvas
    - javafx.scene.canvas.Canvas
- Using a set of graphics commands provided by a GraphicsContext.
  - GraphicsContext
    - javafx.scene.canvas.GraphicsContext

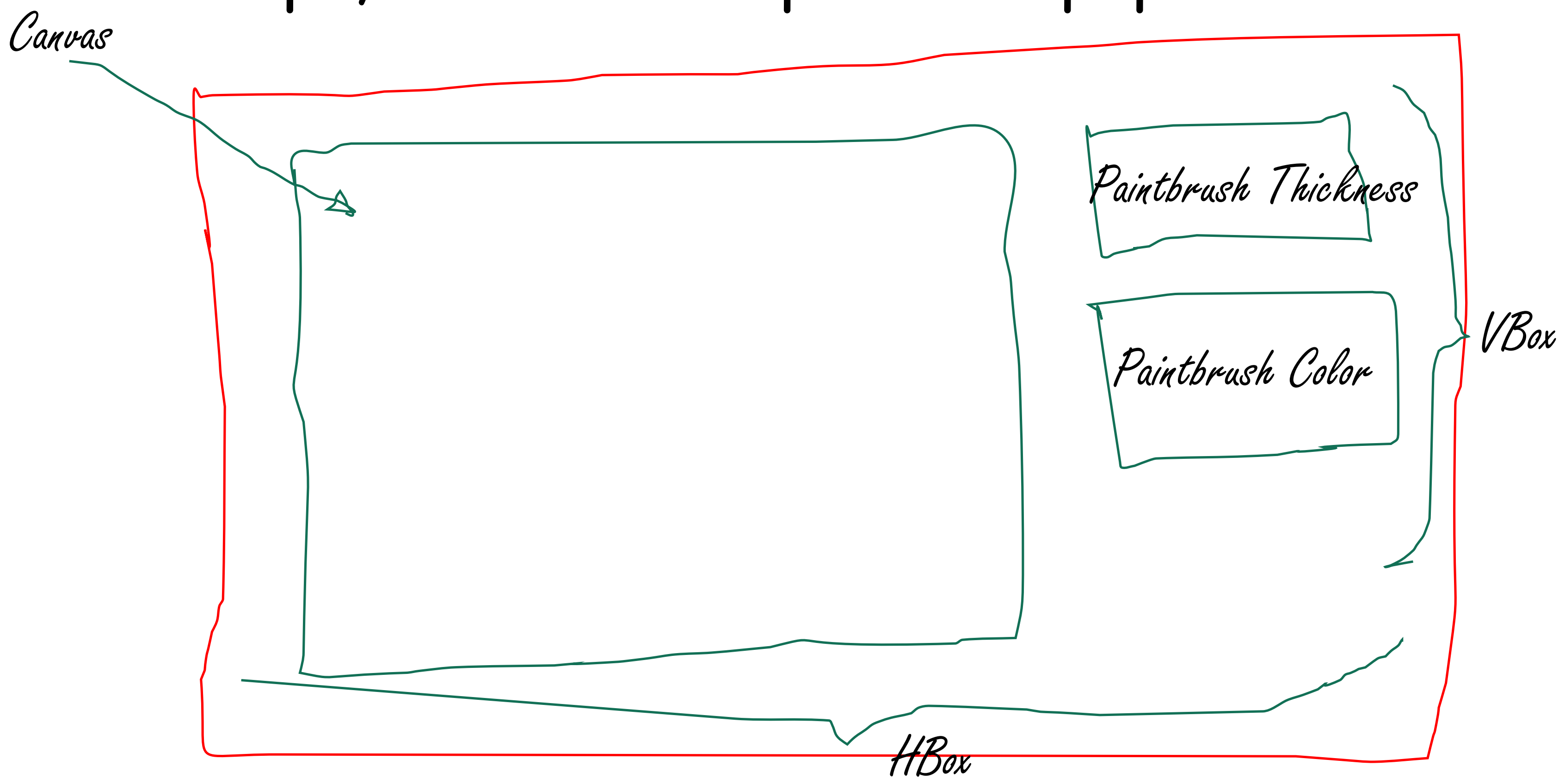
```
Canvas canvas = new Canvas(WIDTH, HEIGHT);
GraphicsContext gc = canvas.getGraphicsContext2D();
```

## Use Build-in UI Controls and Layouts: Example

- Write a JavaFX application with prebuilt UI controls and layouts

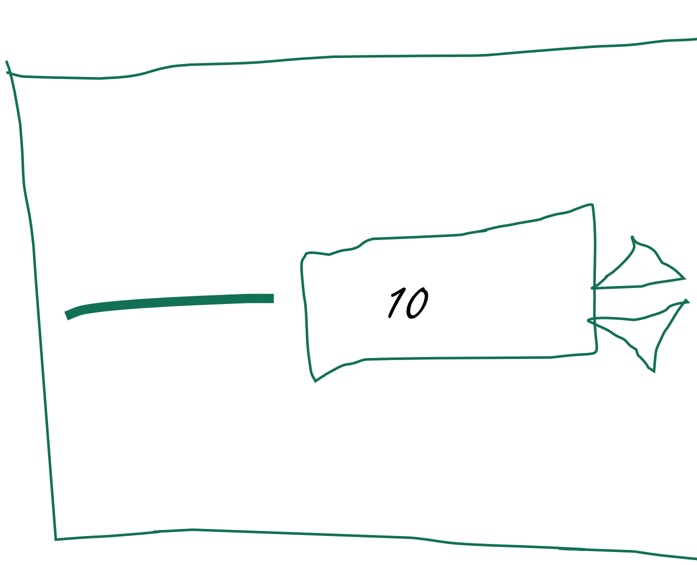
# UI Design: Main Scene

- Perhaps, sketch on a piece of paper



## UI Design: Brush Thickness

- Perhaps, sketch on a piece of paper



# Questions?

- JavaFX build-in components
  - UI controls
  - Text
  - Layouts
  - UI design
- What available in JavaFX?
- Sample applications for exploring JavaFX features
- Assignments

## Explore JavaFX

- The applications are in the “Sample Programs” repository
  - JavaFX Ensemble 8
  - Modena
  - MandelbrotSet
  - 3D Viewer
- In addition to build-in UI controls and layouts, you should explore the following features …

## Assignment

- Practice assignment
- How is Project 2 going?