Advanced Programming Methods

Lecture 9 - JavaFX

Our slides use examples from the followings:

JavaFX tutorials

- https://docs.oracle.com/javase/8/javase-clienttechnologies.htm
- https://wiki.eclipse.org/Efxclipse/Tutorials
- http://o7planning.org/en/11009/javafx
- http://code.makery.ch/library/javafx-8-tutorial/

JavaFX API

- https://docs.oracle.com/javase/8/javafx/api/toc.htm

CONTENT

- What is JavaFX
- JavaFx Architecture
- Steps to install and set JavaFx (on Eclipse)
- Scene graph
- Simple JavaFx Application
- Layout management
- Event Driven Programming – Event Handling
- A Simple Application without SceneBuilder

WHAT IS JAVAFX ?

- Classes and interfaces that provide support for creating Java applications that can be designed, implemented, tested on different platforms.

- Provides support for the use of Web components such as HTML5 code or JavaScript scripts

- Contains graphic UI components for creating graphical interfaces and manage their appearance through CSS files

- Provides support for interactive 3D graphics

- Provides support for handling multimedia content

- Supports RIAs (Rich Internet Application)

- Portability: desktop, browser, mobile, TV, game consoles, Blu-ray, etc.

- Ensures interoperability to Swing

WHAT IS JAVAFX ?

- Java 8 JavaFX is bundled with the Java platform, so JavaFX is available everywhere Java is

- From Java 8 you can also create standalone install packages for Windows, Mac and Linux with Java, which includes the JRE needed to run them.

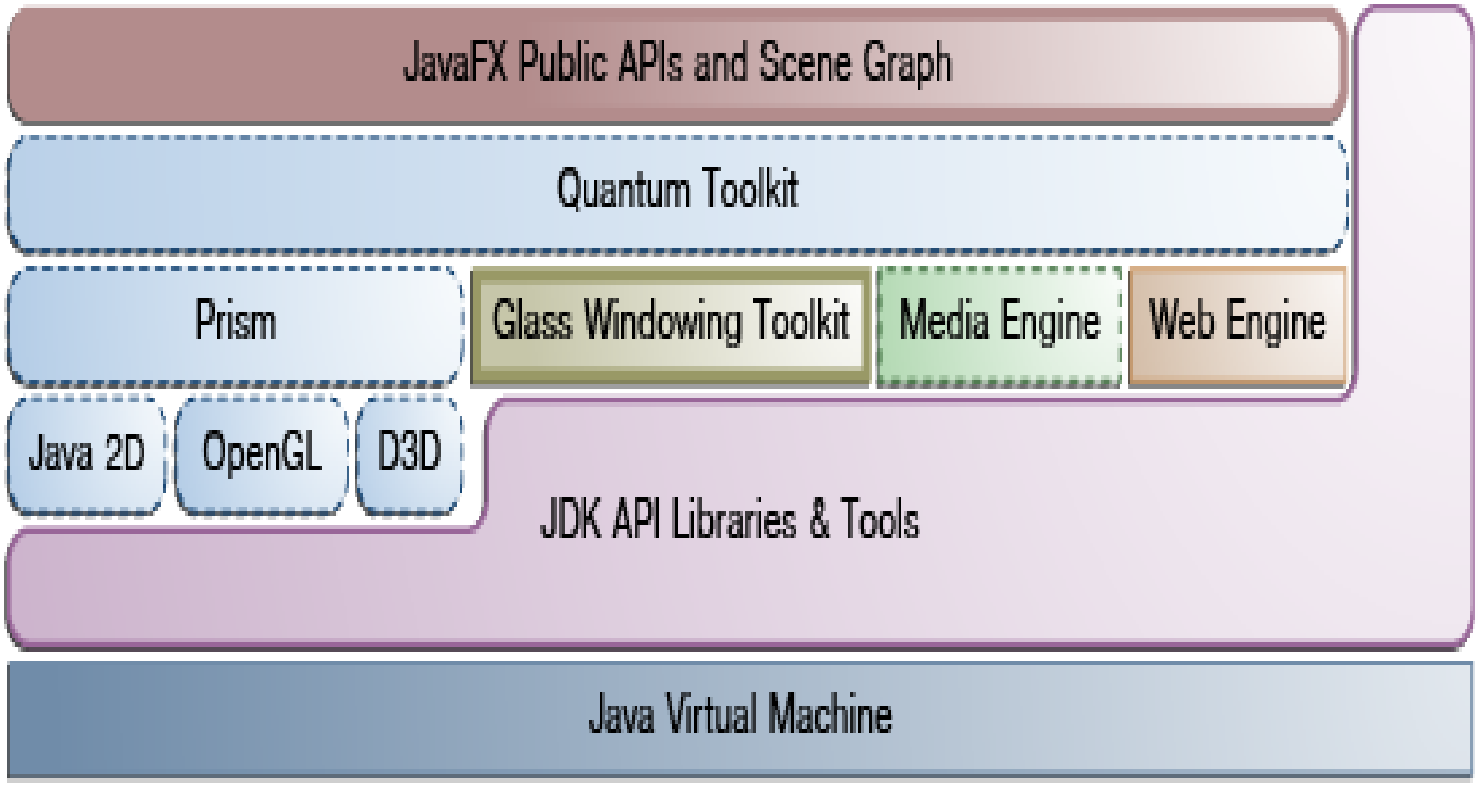
  – This means that you can distribute JavaFX applications to these platforms even if the platform does not have Java installed already

JAVAFX VS. SWING

- JavaFX is intended to replace Swing as the default GUI toolkit in Java.

- JavaFX is more consistent in its design than Swing, and has more features:
  - It is more modern too, enabling you to design GUI using layout files (XML) and style them with CSS, just like we are used to with web applications.
  - JavaFX also integrates 2D + 3D graphics, charts, audio, video, and embedded web applications into one coherent GUI toolkit.

JAVAFX ARCHITECTURE



JAVAFX ARCHITECTURE

- Scene Graph:
  - is the starting point for constructing a JavaFX application.
  - is a hierarchical tree of nodes that represents all of the visual elements of the application's user interface.
  - can handle input and can be rendered.

- Java Public API:
  - provides a complete set of Java public APIs that support rich client application development.

JAVAFX ARCHITECTURE

- Graphics System:

–**Prism:**

•processes render jobs.

•can run on both hardware and software renderers, including 3-D.

•is responsible for rasterization and rendering of JavaFX scenes.

–**Quantum Toolkit**

•ties Prism and Glass Windowing Toolkit together and makes them available to the JavaFX layer above them in the stack.

•also manages the threading rules related to rendering versus events handling.

JAVAFX ARCHITECTURE

- Glass Windowing Toolkit
  - provides native operating services, such as managing the windows, timers, and surfaces.
  - serves as the platform-dependent layer that connects the JavaFX platform to the native operating system.
  - is also responsible for managing the event queue.
  - runs on the same thread as the JavaFX application

JAVAFX ARCHITECTURE

- Running Threads

–**JavaFX application thread:**

•primary thread used by JavaFX application developers

•Any ”live” scene, which is a scene that is part of a window, must be accessed from this thread

•A scene graph can be created and manipulated in a background thread, but when its root node is attached to any live object in the scene, that scene graph must be accessed from the JavaFX application thread

–**Prism render thread:**

•handles the rendering separately from the event dispatcher.

–**Media thread:**

•runs in the background and synchronizes the latest frames through the scene graph by using the JavaFX application thread.

JAVAFX ARCHITECTURE

- Pulse:
  - is an event that indicates to the JavaFX scene graph that it is time to synchronize the state of the elements on the scene graph with Prism.
- The Glass Windowing Toolkit is responsible for executing the pulse events
- Media and Images:
  - JavaFX supports both visual and audio media
- Web Component:
  - is a JavaFX UI control, based on Webkit, that provides a Web viewer and full browsing functionality through its API.

JAVAFX ARCHITECTURE

- JavaFX Cascading Style Sheets (CSS)

  – provides the ability to apply customized styling to the user interface of a JavaFX application without changing any of that application's source code.

  – can be applied to any node in the JavaFX scene graph and are applied to the nodes asynchronously.

  – can also be easily assigned to the scene at runtime, allowing an application's appearance to dynamically change.

JAVAFX ARCHITECTURE

- JavaFX UI Controls

  – available through the JavaFX API

  – are built by using nodes in the scene graph.

  – are portable across different platforms

JAVAFX ARCHITECTURE

- Layout:
  - –Layout containers (panes) can be used to allow for flexible and dynamic arrangements of the UI controls within a scene graph
- Transformations 2-D and 3-D
  - –Each node in the JavaFX scene graph can be transformed in the x-y coordinate
- Visual Effects:
  - –to enhance the look of JavaFX applications in real time.
  - –are primarily image pixel-based

STEPS TO INSTALL AND SET JAVAFX (ON ECLIPSE)

- **Step 1:** install e(fx)clipse into Eclipse (JavaFx tooling)
  - You can follow the steps from one of the followings:
  - For the latest Eclipse please use https://marketplace.eclipse.org/content/efxclipse and then follow
    - http://o7planning.org/en/10619/install-efxclipse-into-eclipse
    - https://wiki.eclipse.org/Efxclipse/Tutorials/AddingE(fx)clipse_to_eclipse

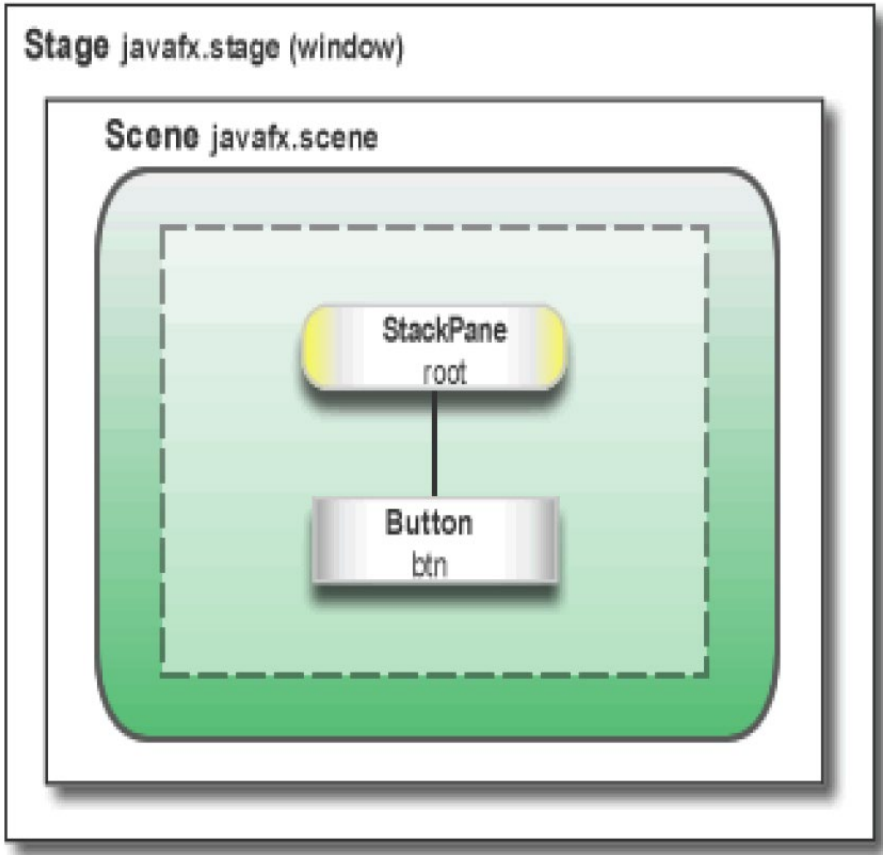
- **Step 2:** download and set JavaFx SceneBuilder
  - You can follow the steps from http://o7planning.org/en/10621/install-javafx-scene-builder-into-eclipse
  - You can download the SceneBuilder
    - Either from Oracle http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-1x-archive-2199384.html
    - Or from http://gluonhq.com/products/scene-builder/

SCENE GRAPH

scene-graph-based programming model

A JavaFX application contains:

- An object **Stage (window)**
- One or more objects **Scene**



SCENE GRAPH

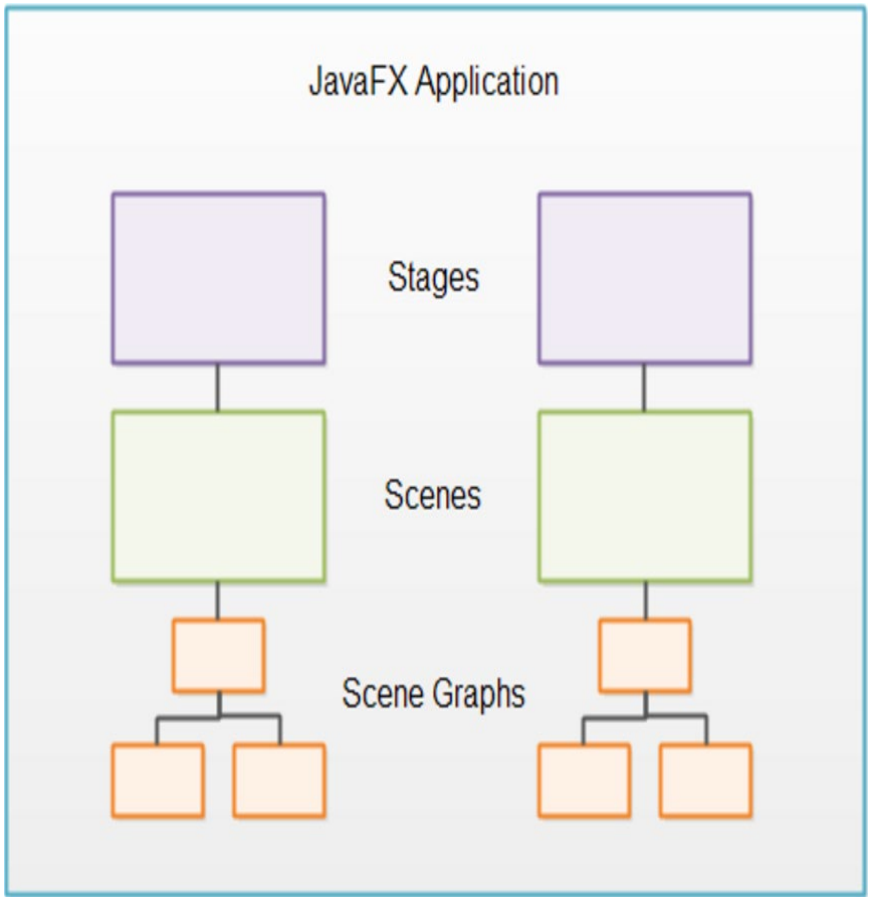
- A Scene Graph is a tree of graphical user interface components.

- A scene graph element is a node.
- Each node has an ID, a class style, a bounding volume, etc.

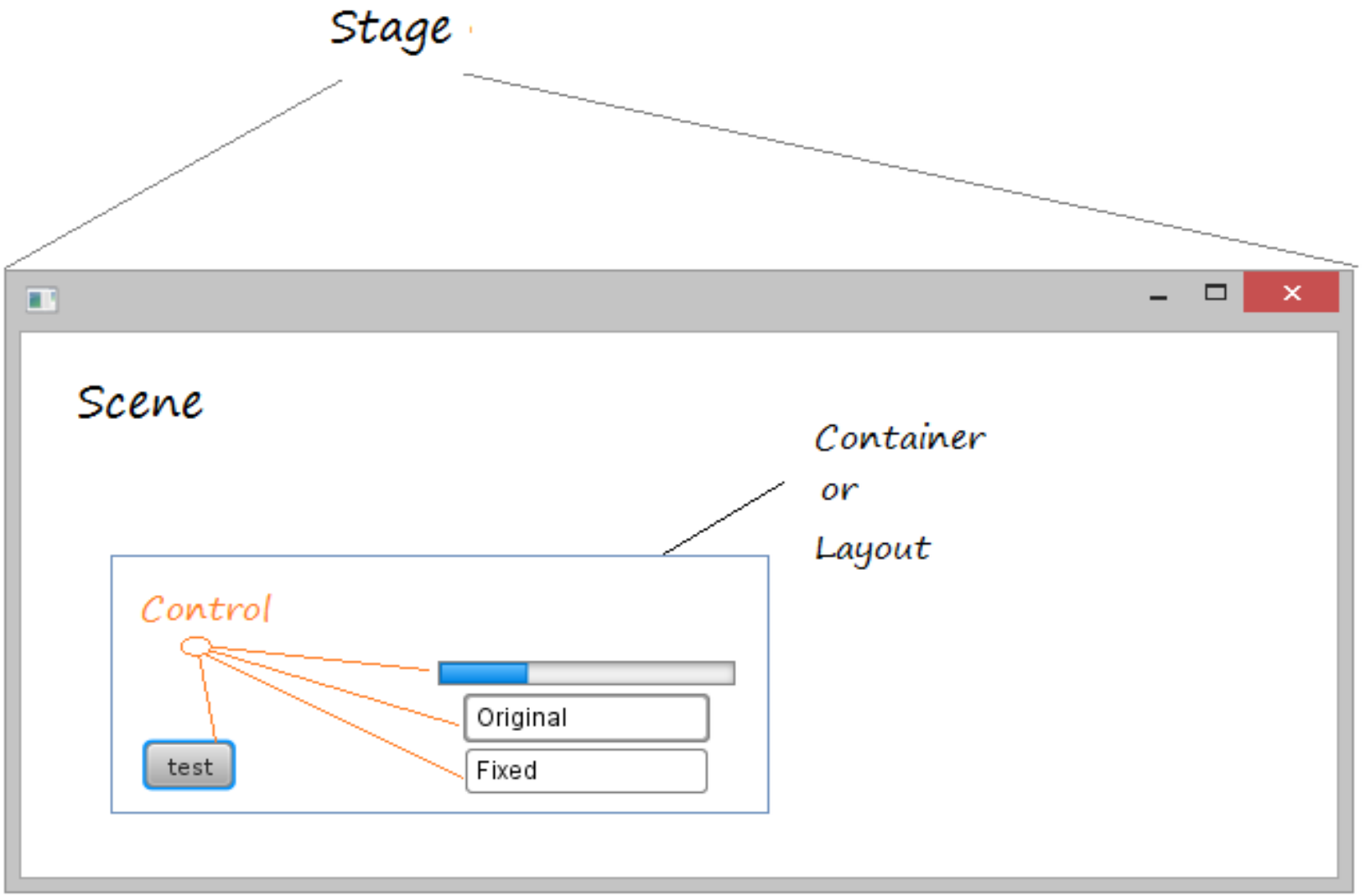
- Except the root node, each node has a single parent and 0 or more children.
- Nodes can be internal (Parent) or leaf

- A node can be associated with various properties (effects (blur, shadow), opacity, transformations) and events (event handlers (Mouse, Keyboard))

SCENE GRAPH



SCENE GRAPH


Stage
Scene
Container
or
Layout
Control
Original
Fixed
test

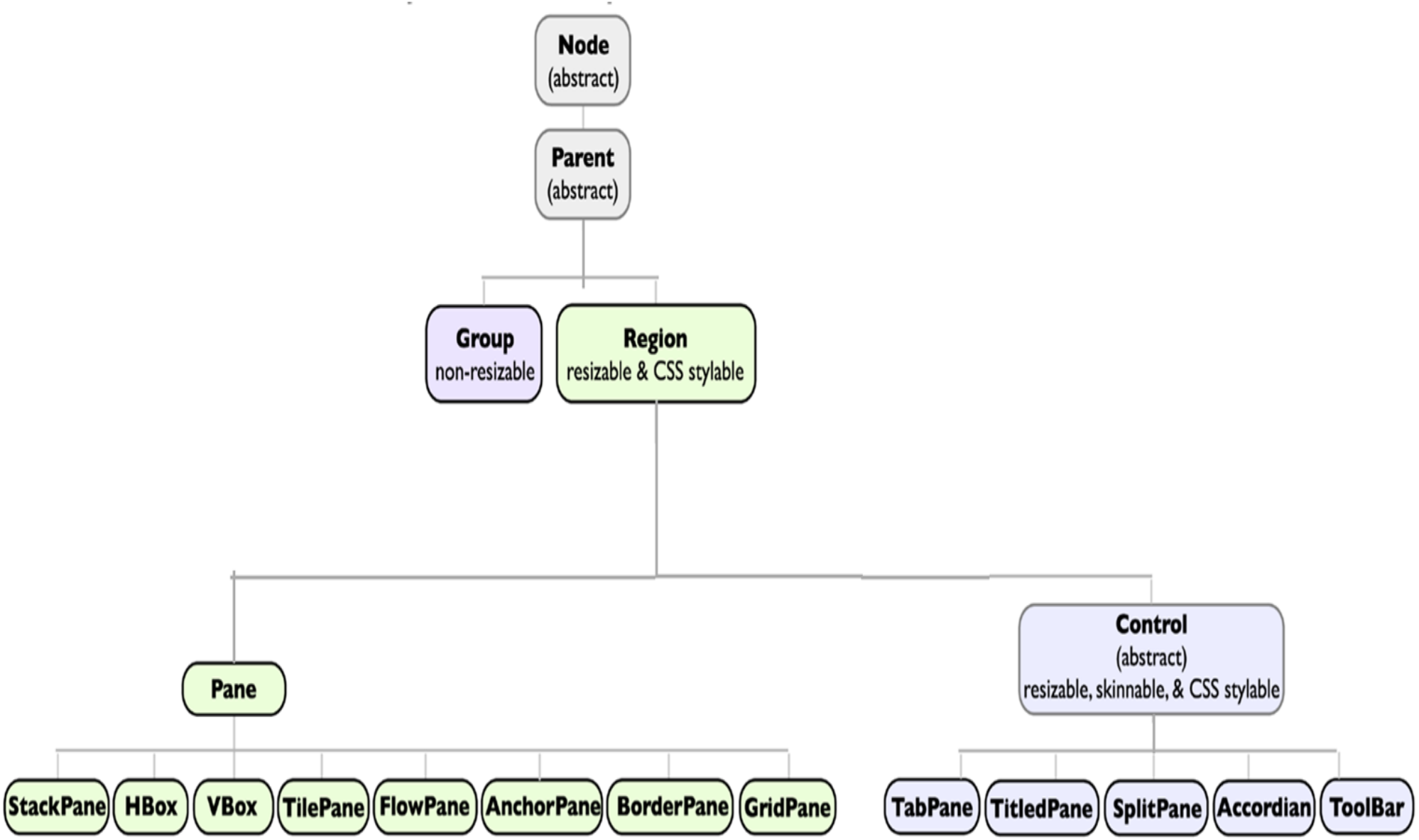
SCENE GRAPH

- Node: The abstract base class for all scene graph nodes.

- Parent: The abstract base class for all branch nodes. (This class directly extends Node).

- Scene: The base container class for all content in the scene graph.

SCENE GRAPH


Node
(abstract)
Parent
(abstract)
Group
non-resizable
Region
resizable & CSS stylable
Pane
Control
(abstract)
resizable, skinnable, & CSS stylable
StackPane
HBox
VBox
TilePane
FlowPane
AnchorPane
BorderPane
GridPane
TabPane
TitledPane
SplitPane
Accordian
ToolBar

JAVAFX APPLICATION

- A JavaFX application is an instance of class **Application**

```
public abstract class Application extends Object;
```

- Creation of a new object of class Application is done by executing the static method ***launch()*** from the class ***Application***:

```
public static void Launch(String... args);
```

- `args` **aplication parameters**(parameters of the method *main*).

- JavaFX runtime executes the following steps:

1. Create a new object Application
2. Call the method init of the new object Application
3. Call the method start of the new object Application
4. Wait for the application to terminate

JAVAFX APPLICATION

- Note that the start method is abstract and must be overridden.
- The init and stop methods have concrete implementations that do nothing.
- Application parameters can be obtained by calling *getParameters*() method from the init() method, or any time after the init method has been called.

JAVAFX APPLICATION THREAD

- JavaFX creates an application thread for running the application start method, processing input events, and running animation timelines.

- Creation of JavaFX Scene and Stage objects as well as modification of scene graph operations to live objects (those objects already attached to a scene) must be done on the JavaFX application thread.

- The Java launcher loads and initializes the specified Application class on the JavaFX Application Thread.

  – If there is no main method in the Application class, or if the main method calls Application.launch(), then an instance of the Application is then constructed on the JavaFX Application Thread.

- The init method is called on the launcher thread, not on the JavaFX Application Thread.

  – This means that an application must not construct a Scene or a Stage in the init method. An application may construct other JavaFX objects in the init method.

JAVAFX APPLICATION

```
public class Main extends Application {

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 500, 500, Color.PINK);
        stage.setTitle("Welcome to JavaFX!");

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

    public static void main(String[] args) {
        launch(args); //an object Application is created
    }
}
```

ADDING NODES

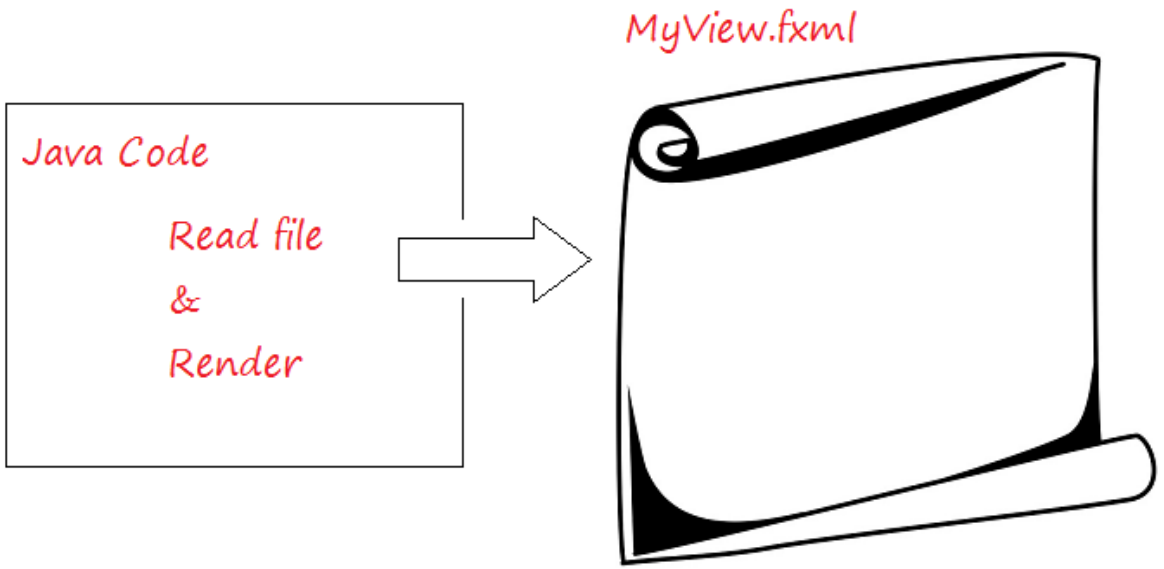
```
// A node of type Group is created
Group group = new Group();


// A node of type Rectangle is created
Rectangle r = new Rectangle(25,25,50,50);
r.setFill(Color.BLUE);
group.getChildren().add(r);

// A node of type Circle is created
Circle c = new Circle(200,200,50, Color.web("blue", 0.5f));
group.getChildren().add(c);
```

JAVAFX APPLICATION

DEMO

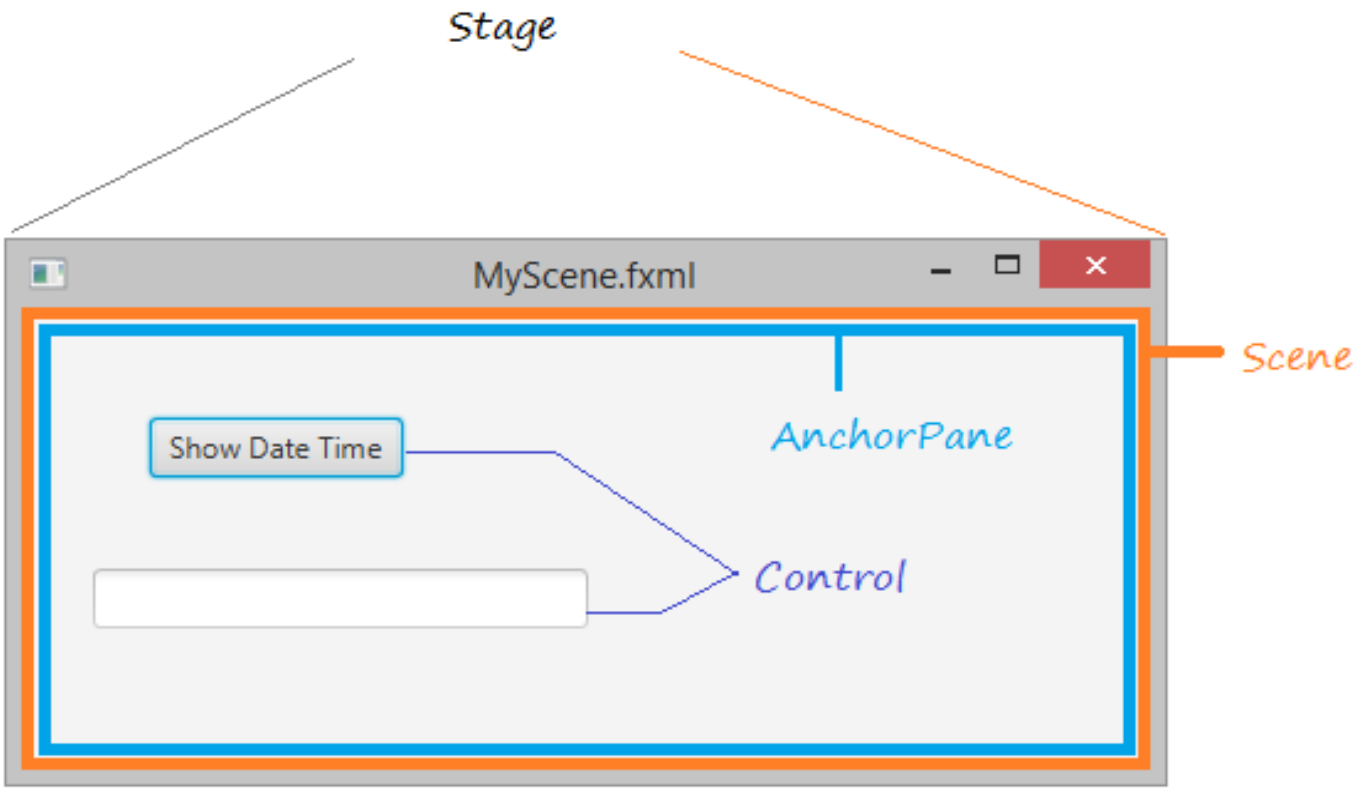
JAVAFX SCENE BUILDER

- to create a JavaFX application interface, you can write code in Java entirely.
- it takes so much time to do this,
- JavaFX Scene Builder is a visual tool allowing you to design the interface of Scene.
- The code which is generated, is XML code saved on *.fxml file.



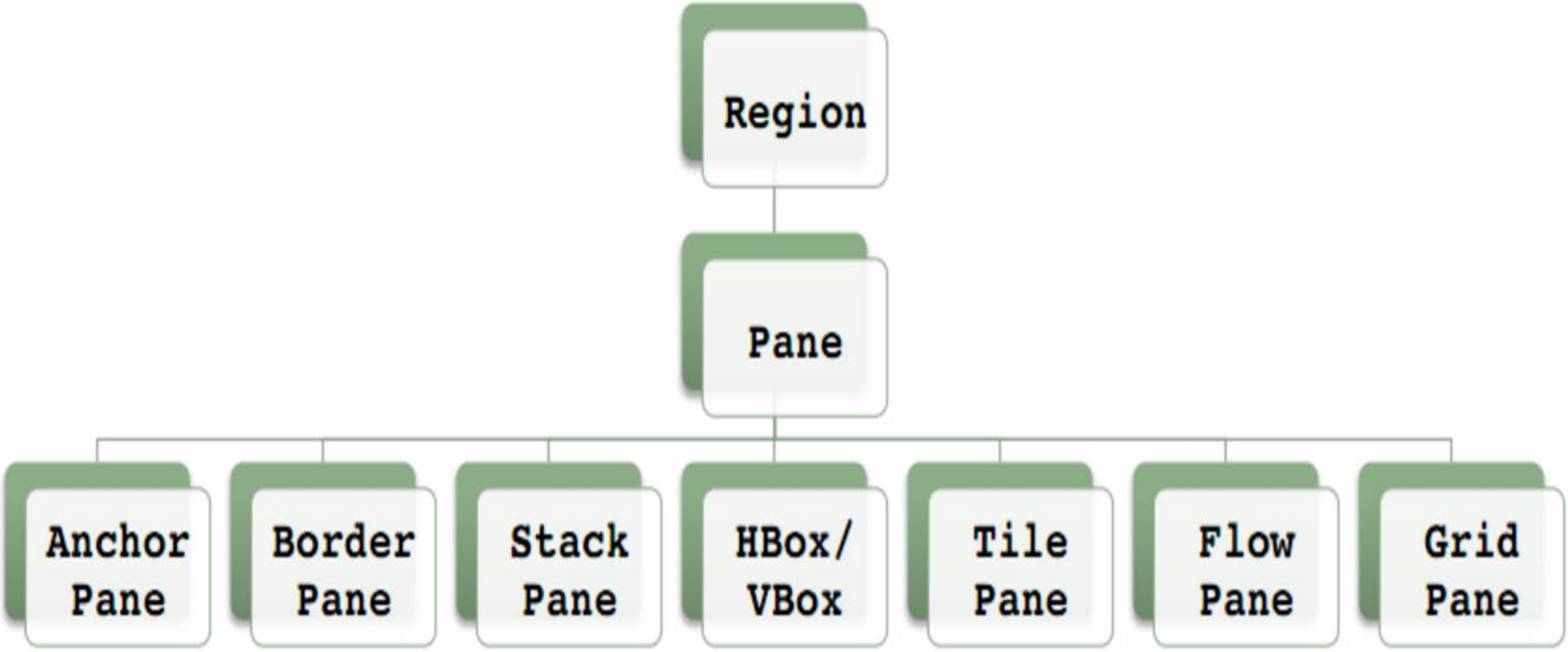
JAVAFX SCENE BUILDER

.DEMO

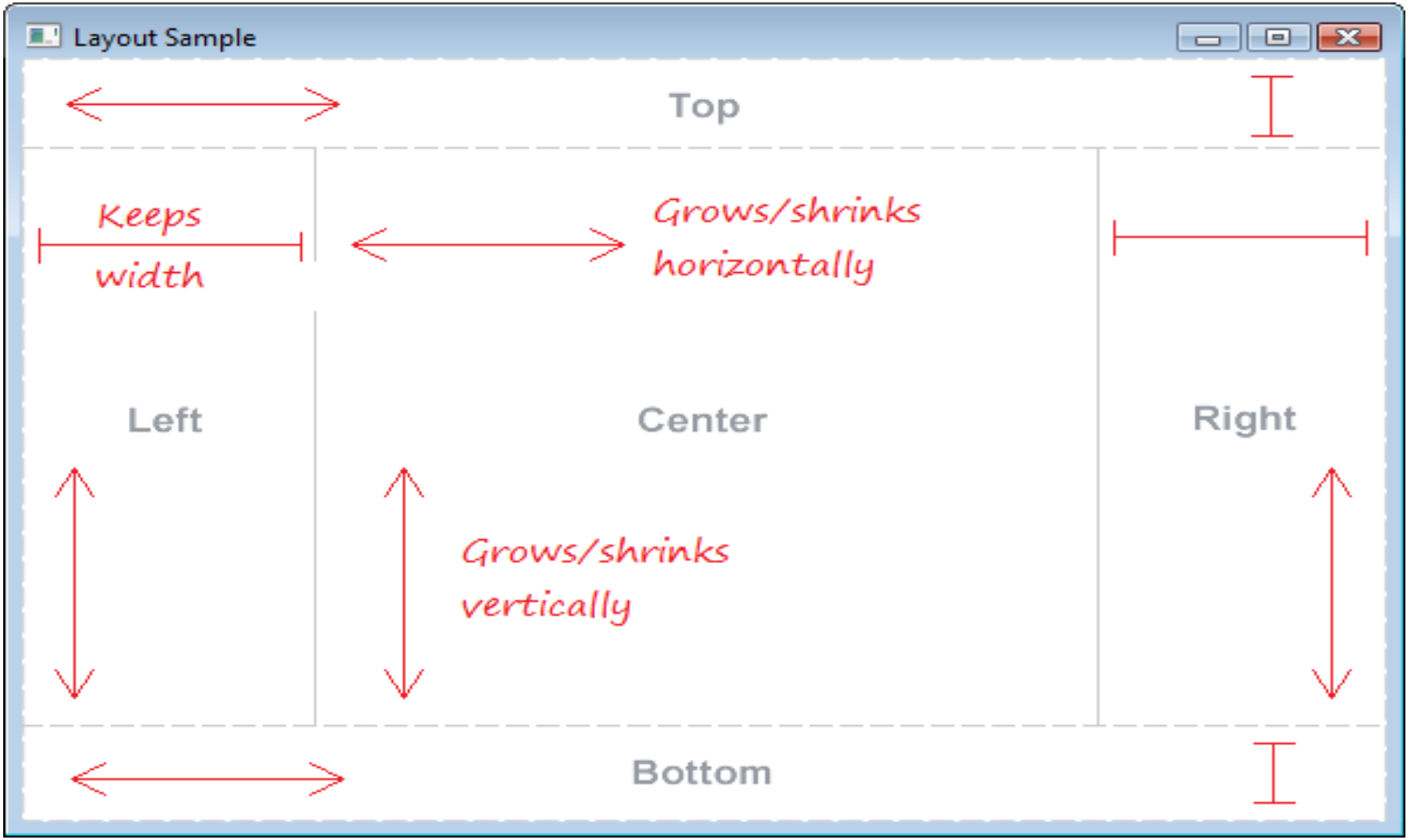


LAYOUT MANAGEMENT

- layouts are components which contains other components inside them and manage the nested components

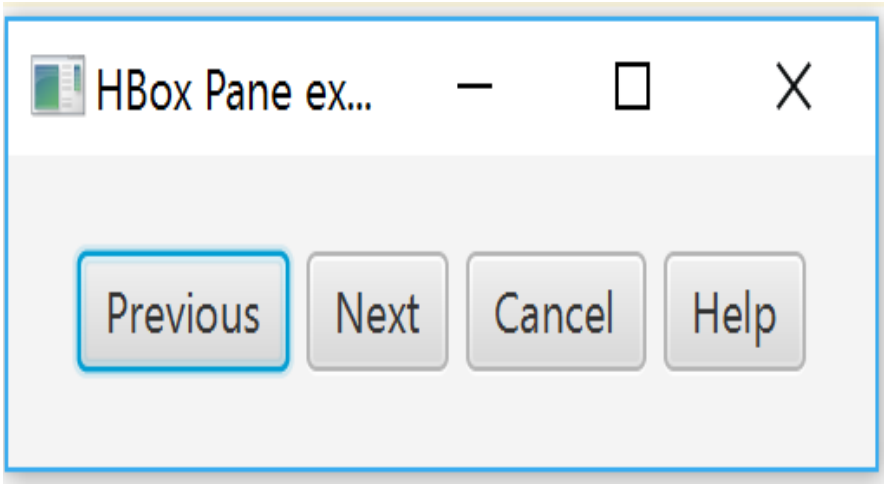


LAYOUT MANAGEMENT - BORDERPANE



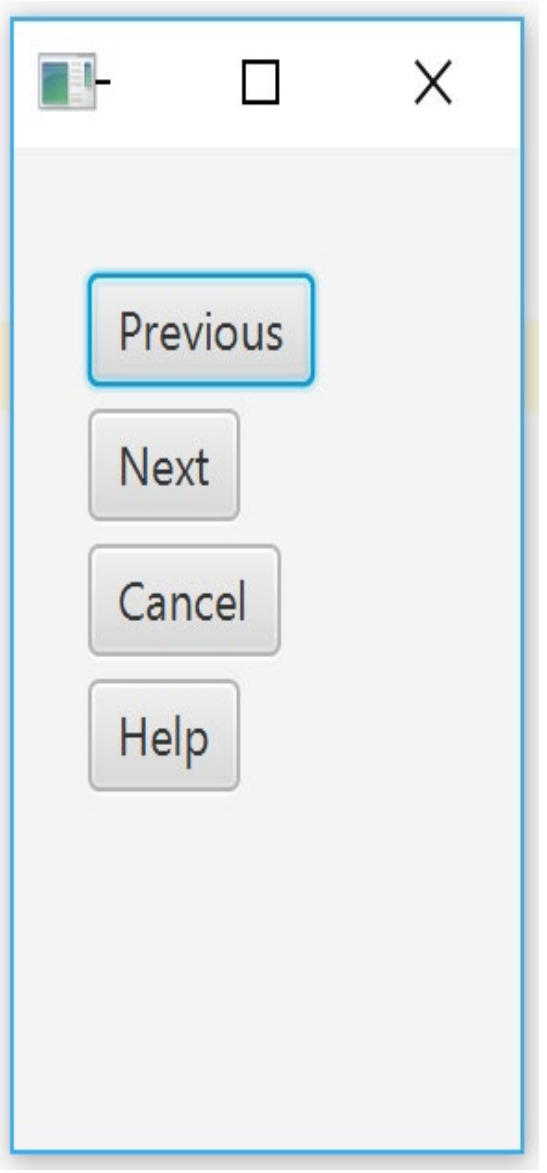
LAYOUT MANAGEMENT - HBOX

```
HBox root = new HBox(5);
root.setPadding(new Insets(100));
root.setAlignment(Pos.BASELINE_RIGHT);

Button prevBtn = new Button("Previous");
Button nextBtn = new Button("Next");
Button cancBtn = new Button("Cancel");
Button helpBtn = new Button("Help");

root.getChildren().addAll(prevBtn,
nextBtn, cancBtn, helpBtn);
```



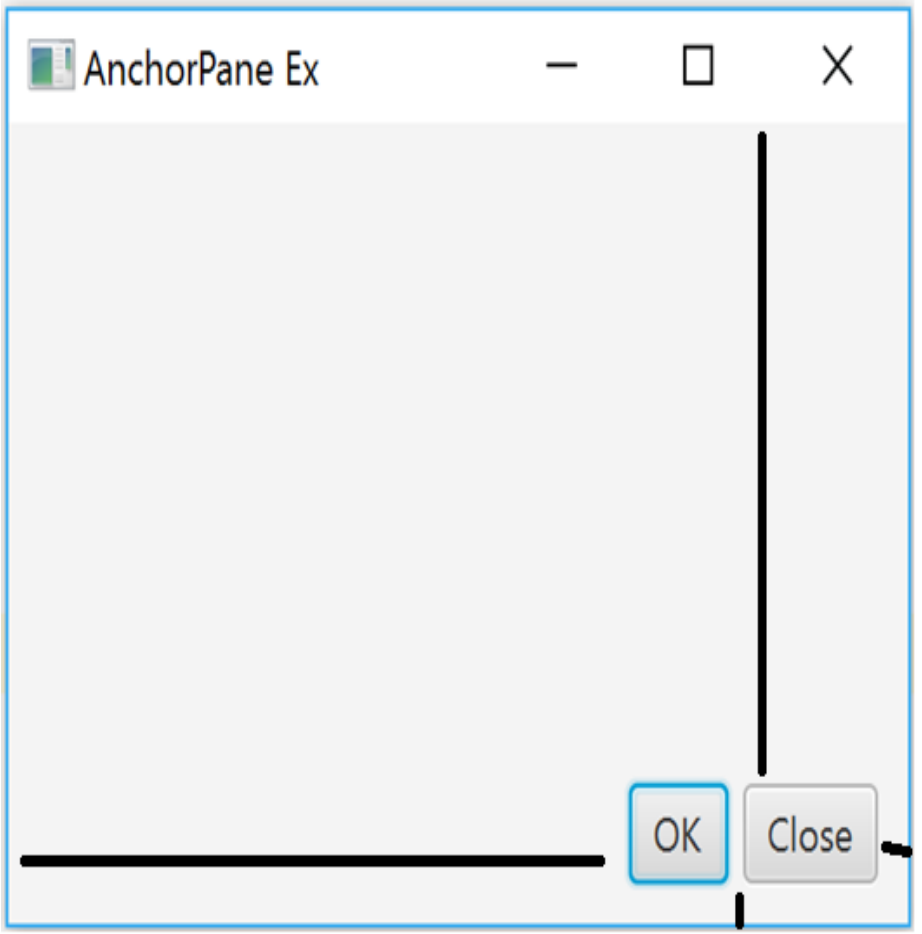
LAYOUT MANAGEMENT - VBOX

```
VBox root = new VBox(5);
root.setPadding(new Insets(20));
root.setAlignment(Pos.BASELINE_LEFT);

Button prevBtn = new Button("Previous");
Button nextBtn = new Button("Next");
Button cancBtn = new Button("Cancel");
Button helpBtn = new Button("Help");

root.getChildren().addAll(prevBtn,
nextBtn, cancBtn, helpBtn);
Scene scene = new Scene(root, 150, 200);
```



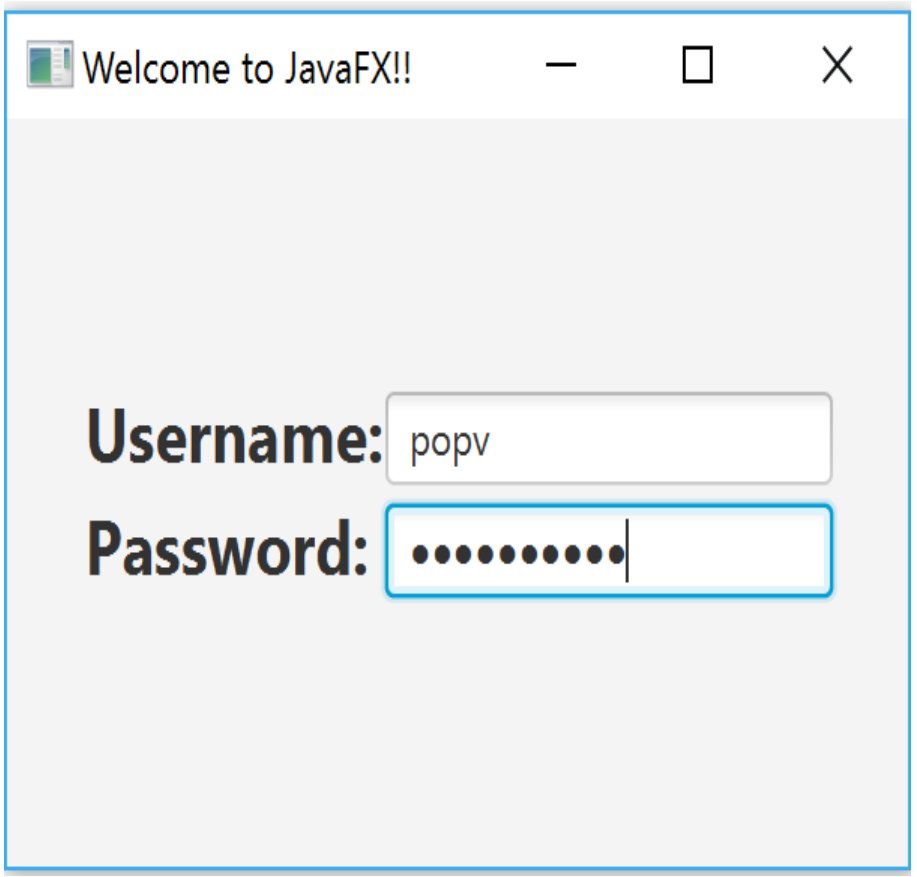
LAYOUT MANAGEMENT – ANCHORPANE

```
AnchorPane root = new AnchorPane();

Button okBtn = new Button("OK");
Button closeBtn = new Button("Close");
HBox hbox = new HBox(5, okBtn, closeBtn);

root.getChildren().addAll(hbox);

AnchorPane.setRightAnchor(hbox, 10d);
AnchorPane.setBottomAnchor(hbox, 10d);

Scene scene = new Scene(root, 300, 200);

stage.setTitle("AnchorPane Ex");
stage.setScene(scene);
stage.show();
```



LAYOUT MANAGEMENT - GRIDPANE

```
GridPane gr=new GridPane();
gr.setPadding(new Insets(20));
gr.setAlignment(Pos.CENTER);

gr.add(createLabel("Username:"),0,0);
gr.add(createLabel("Password:"),0,1);

gr.add(new TextField(),1,0);
gr.add(new PasswordField(),1,1);

Scene scene = new Scene(gr, 300, 200);
stage.setTitle("Welcome to JavaFX!!");
stage.setScene(scene);
stage.show();
```



JAVAFX CONTROLS

- Are the main components of the GUI
- A control is a node in the scene graph
- Can be manipulated by the user
- Java FX Controls reference:

https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336

Button Radio Button Toggle Button Checkbox Choice Box Text FieldLabel Password Field Scroll Bar Scroll Pane	List View Table View Tree View Combo Box Separator Slider Progress Bar and Progress Indicator Hyperlink	Tooltip HTML Editor Titled Pane and Accordion Menu Color Picker Pagination Control File Chooser Customization of UI Controls

EVENT DRIVEN PROGRAMMING

Event: Any user action generates an event:

–pressing or releasing the keyboard,

–moving the mouse,

–pressing or releasing a button mouse,

–opening or closing a window,

–performing a mouse click on a component of the interface,

–entering / leaving the mouse cursor in/out of a component area

–...

- There are also events that are not generated by the application user.
- An event can be treated by executing a program module.

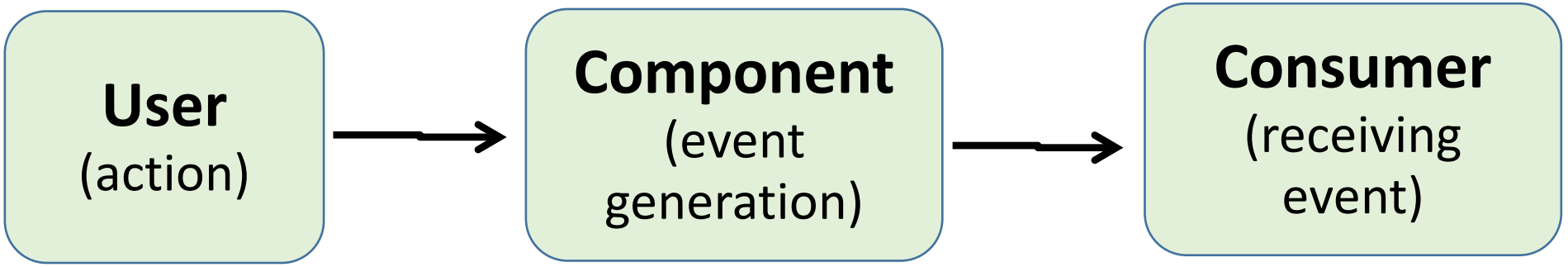
EVENTS HANDLING

- **Delegation Event Model**

We can distinguish three categories of objects used to handle events:

- **Events Sources** - those objects that generate the events;
- **Events** -which are objects (generated by sources and received by consumers)
- **Events consumers or listeners** - those objects that receive and treat the events.

EVENTS HANDLING

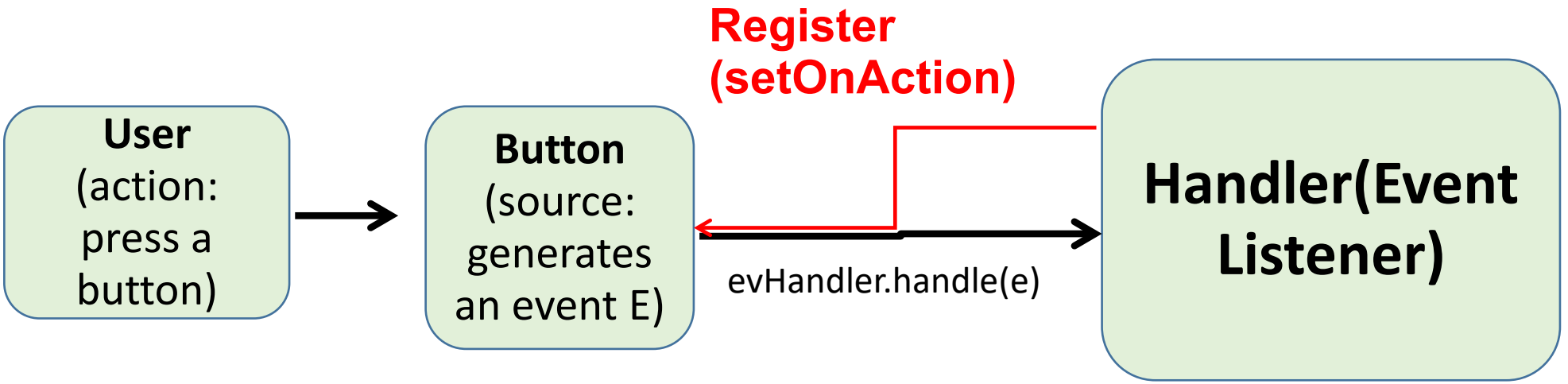


Each consumer must be registered with the event source. This process ensures that the source knows all the consumers to which must submit its generated events.

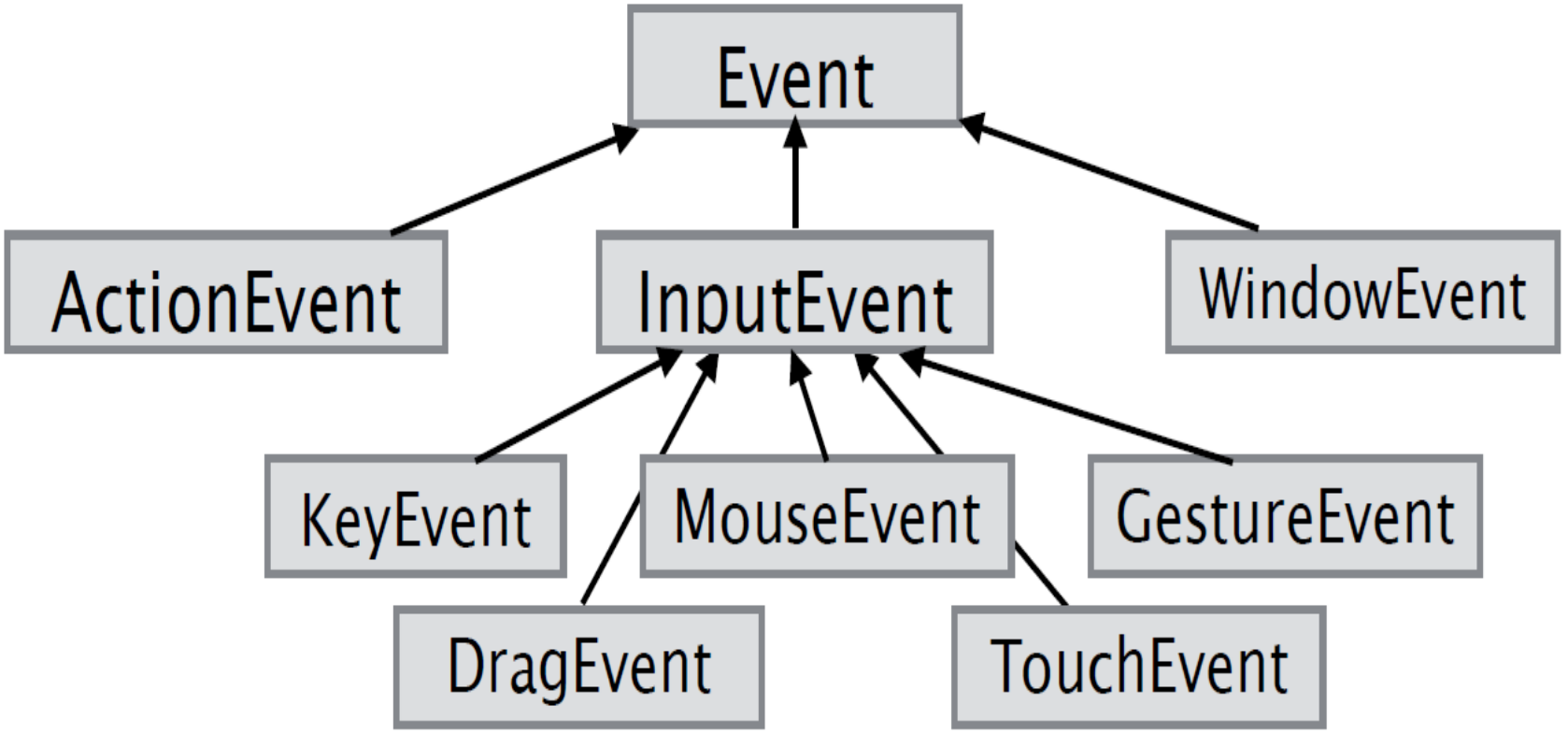
The "delegation" assumes that an event source (an object) transmits all its generated events to those consumers which were recorded at it.

A consumer receives events only from those sources to which it have been registered !!!

EVENTS HANDLING



TYPES OF EVENTS



EVENT HANDLER

```
@FunctionalInterface
Interface EventHandler<T extends Event> extends
EventListener{
        void   handle(T event);
}
```

EVENT HANDLER

Button Events

Only one handler can be associated to the button clicked event!!!

```
Button btn = new Button("Ding!");
btn.setStyle("-fx-font: 42 arial; -fx-base: #f8e7f9;");
// handle the button clicked event
btn.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {
        System.out.println("Hello World!");
    }
});
```

Or using lambda expressions:

```
btn.setOnAction(e->System.out.println("Hello World!"));
```

See HelloWord.zip

A SIMPLE APPLICATION

- We follow the Oracle tutorial to create manually (without SceneBuilder) a simple form JavaFx application

- **See Ex1-NoSceneBuilder.zip**

-

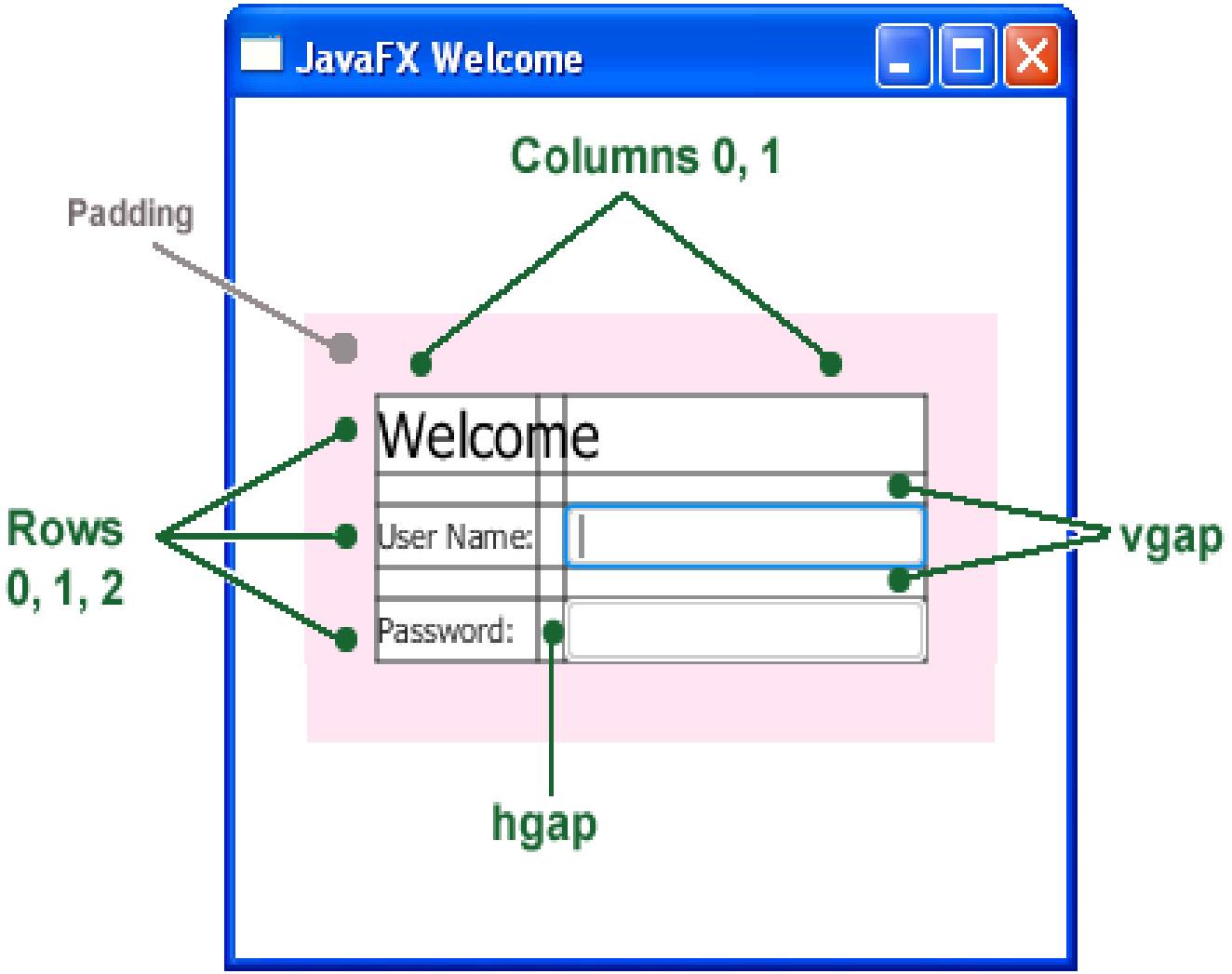
CREATE A GRIDPANE WITH GAP AND PADDING PROPERTIES

```
public void start(Stage primaryStage) {

Try {

primaryStage.setTitle("JavaFX Welcome");

GridPane grid = new GridPane();

grid.setAlignment(Pos.CENTER);

grid.setHgap(10);

grid.setVgap(10);

grid.setPadding(new Insets(25, 25, 25, 25));

.......

Scene scene = new Scene(grid,400,400);

scene.getStylesheets().add(getClass().getResource("application.cs
s").toExternalForm());

primaryStage.setScene(scene);

primaryStage.show();

} catch(Exception e) {e.printStackTrace();}}
```

ADD TEXT, LABELS, AND TEXT FIELDS

```
Text scenetitle = new Text("Welcome");
scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
grid.add(scenetitle, 0, 0, 2, 1);

Label userName = new Label("User Name:");
grid.add(userName, 0, 1);

TextField userTextField = new TextField();
grid.add(userTextField, 1, 1);
Label pw = new Label("Password:");
grid.add(pw, 0, 2);
PasswordField pwBox = new PasswordField();
grid.add(pwBox, 1, 2);
```

ADD TEXT, LABELS, AND TEXT FIELDS



ADD A BUTTON AND TEXT

```
Button btn = new Button("Sign in");

HBox hbBtn = new HBox(10);

hbBtn.setAlignment(Pos.BOTTOM_RIGHT);

hbBtn.getChildren().add(btn);

grid.add(hbBtn, 1, 4);

final Text actiontarget = new Text();

grid.add(actiontarget, 1, 6);
```

STYLLING A BUTTON

```
btn.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9;");
```

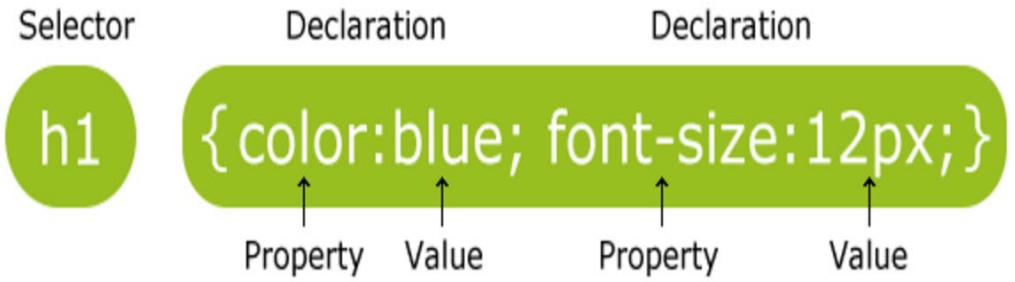
ADD A CASCADING STYLE SHEET (CSS)

- Create a .css file and apply the new styles on the previous example
- By switching to CSS over inline styles, we separate the design from the content.
- This approach makes it easier for a designer to have control over the style without having to modify content.
- JavaFx CSS Reference Guide: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

- First: Initialize the stylesheets Variable

```
scene.getStylesheets().add(getClass().getResource("application.css"
).toExternalForm());
```

CSS



http://www.w3schools.com/css/

```
.button {
    -fx-padding: 5 22 5 22;
    -fx-border-color: #e2e2e2;
    -fx-border-width: 2;
    -fx-background-radius: 0;
    -fx-background-color: #1e2e2e2;
    -fx-font-family: "Segoe UI", Helvetica, Arial,
sans-serif;
    -fx-font-size: 11pt;
    -fx-text-fill: black;
    -fx-background-insets: 0 0 0 0, 0, 1, 2;
}

.button:hover {
    -fx-background-color: #3a3a3a;
}
```

```
.background {
    -fx-background-color:
#1d1d1d;
}

.label {
    -fx-font-size: 11pt;
    -fx-font-family: "Segoe UI
Semibold";
    -fx-text-fill: white;
    -fx-opacity: 0.6;
}
```

ADD A BACKGROUND IMAGE

- The background image is applied to the .root style, which means it is applied to the root node of the Scene instance.

- The style definition consists of the name of the property (-fx-background-image) and the value for the property (url(”backgr.jpg”)).

```
.root {

      -fx-background-image: url("backgr.jpg");

}
```

STYLE THE LABELS

- the .label style class will affect all labels in the form

```
.label {
    -fx-font-size: 12px;
    -fx-font-weight: bold;
    -fx-text-fill: #333333;
    -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) ,
0,0,0,1 );
}
```

STYLE TEXT

- Remove code that define the inline styles

```
//                  scenetitle.setFont(Font.font("Tahoma",
FontWeight.NORMAL, 20));

//                  btn.setStyle("-fx-font: 22 arial; -fx-base:
#b6e7c9;");

//              actiontarget.setFill(Color.FIREBRICK);
```

- Create an ID for each text node by using the setID() method of the Node class

```
scenetitle.setId("welcome-text");

actiontarget.setId("actiontarget");
```

STYLE TEXT

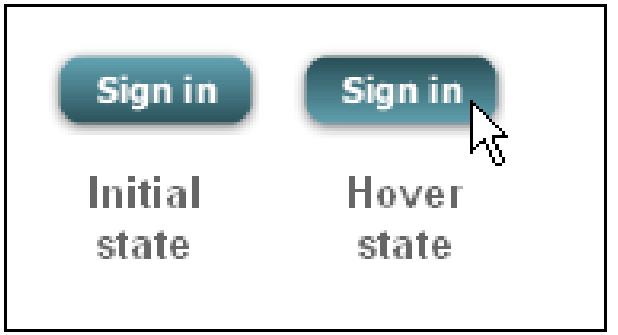
•Add to CSS file

```
#welcome-text {
    -fx-font-size: 32px;
    -fx-font-family: "Arial Black";
    -fx-fill: #818181;
    -fx-effect: innershadow( three-pass-box , rgba(0,0,0,0.7) , 6, 0.0 , 0 , 2 );
}
#actiontarget {
   -fx-fill: FIREBRICK;
   -fx-font-weight: bold;
   -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );
}
```

STYLE TEXT

- Text with shadow effects

STYLE THE BUTTON



•Initial state

```
.button {
    -fx-text-fill: white;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold;
    -fx-background-color: linear-gradient(#61a2b1, #2A5058);
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );
}
```

•Hover state

```
.button:hover {
    -fx-background-color: linear-gradient(#2A5058, #61a2b1);
}
```