# Programming of Interactive Systems

Anastasia.Bezerianos@lri.fr



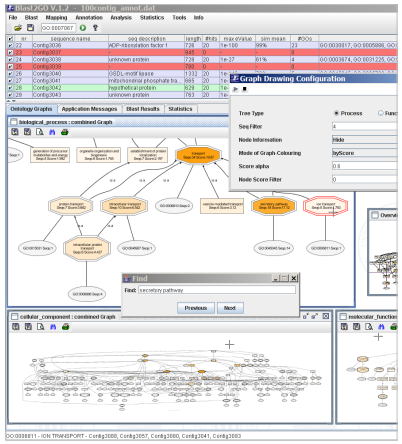
# Week 2 :
# a. Intro to JavaFX

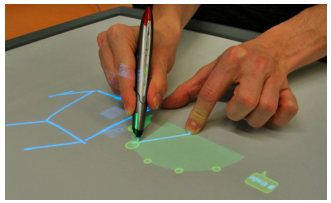
Anastasia.Bezerianos@lri.fr

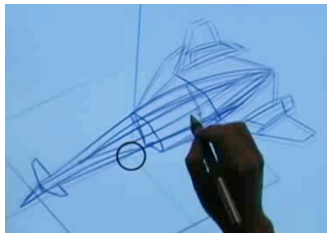
(part of this class is based on previous classes from Anastasia,
and of T. Tsandilas, S. Huot, M. Beaudouin-Lafon, N.Roussel, O.Chapuis)



# interactive systems









# graphical interfaces

A **graphical** **user interface** or **GUI**, is an interface that allows users to interact with electronic devices through graphical icons and visual components (widgets)

**GUIs:** input events (and their result) are specified w.r.t. output (on what widget they act on)

# events

**Event:** An object that represents a user's interaction with a GUI component; can be "handled" to create interactive components

Types of UI events:

- Mouse: move/drag/click  button press/release, …
- Keyboard: key press/release sometimes with modifiers like shift/control/alt, …
- Touchscreen: finger tap/drag, …
- Window: resize/minimize/restore/close, etc.



# interface toolkits

libraries of interactive objects (« widgets », e.g. buttons) that we use to construct interfaces

functions to help programming of GUIs

usually also handle input events

This week we focus on a specific toolkit, JavaFX. Later in the class we will discuss toolkits more …



# JavaFX



# JavaFX toolkit (and ancestors)

- Original Java GUI was the Abstract Window Toolkit (AWT), that was platform dependent.
- Swing was introduced in 1997 to fix the problems with AWT, with higher level components, with pluggable look and feel.
- Swing is built on AWT, default until Java 7 (likely will never die, many Apps running it)
- Since Java 8, **JavaFX** is included in SDK but still not the standard (built on Swing/AWT)

# JavaFX

Java + Flash + Flex

As with Java, it is cross-platform

Can use tools for interface building WYSIWYG (SceneBuilder, more later)

Supports advanced event handling (Swing/AWT)

CSS styling



# JavaFX

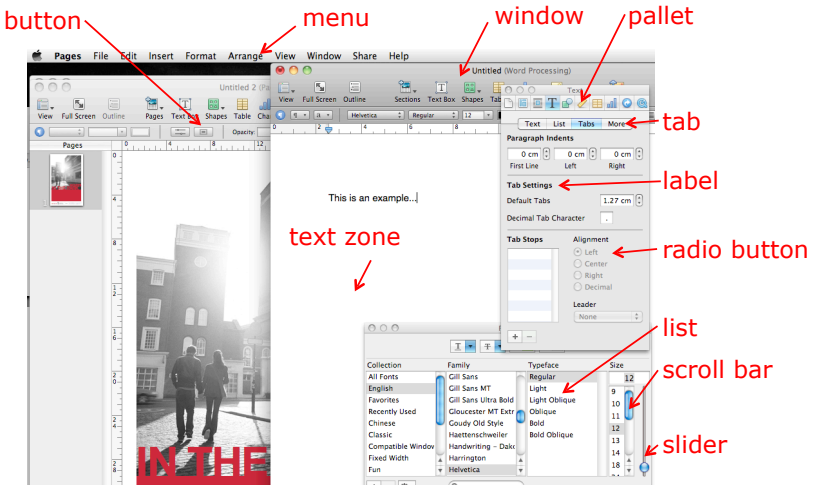
Minimum development tools:
(check with your TA !!!)

- IDE (Eclipse, Netbeans, JDeveloper Oracle)
- JDK 8 or higher
- JDK 8 → Java FX 2.2 (separate installer)
- JDK 8 → JAVA FX 8 (included in SDK installer)
- (SceneBuilder 1.0 or newer)

(note) if eclipse gives you an "Access Restriction" error, go to Project→Properties→Java Build Path→Libraries open JRE, edit Access rules and Add: Accessible javafx/**

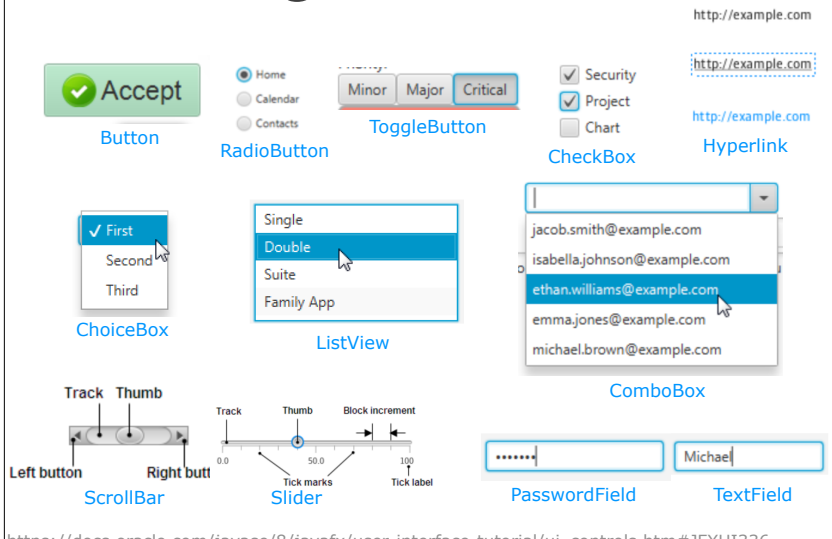


# « widgets » (window gadget)



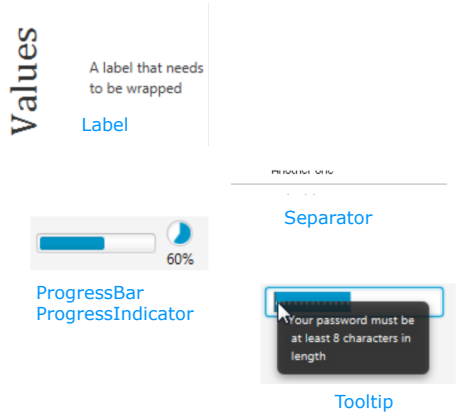



# JavaFX widgets (atomic interactive)




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

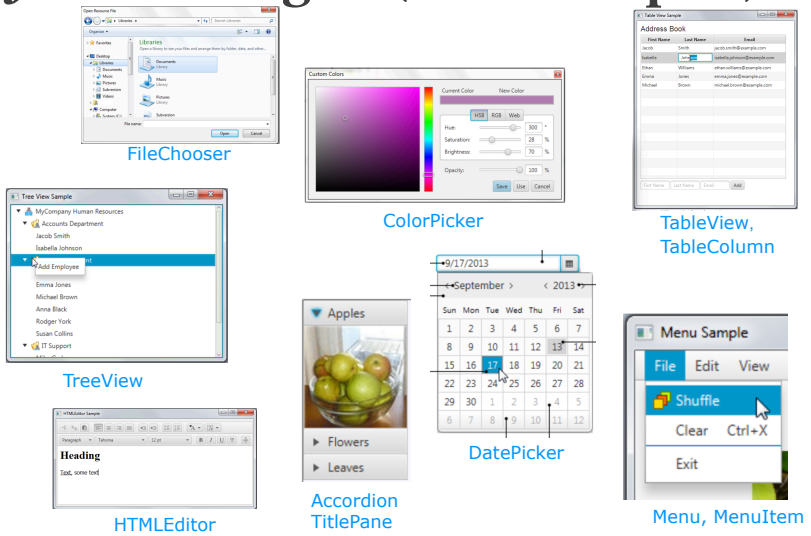
# JavaFX widgets (non-editable)




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



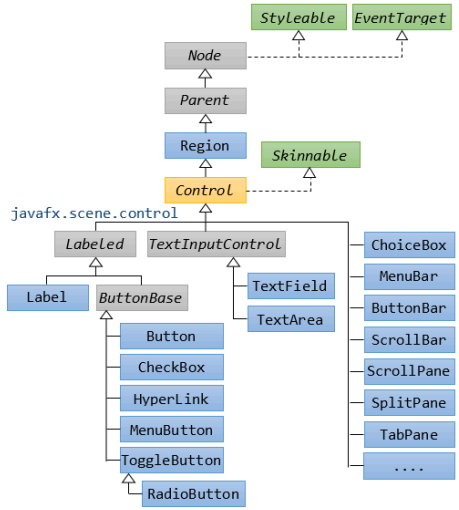
# JavaFX widgets (more complex)




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



# JavaFX widget (control) classes




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



# JavaFX - our first window

*New Project → New Class (HelloWorld)*

```
import javafx.application.Application;
import javafx.stage.Stage;

public class HelloWorld extends Application {

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

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("First GUI window!");
        primaryStage.show();
    }

}
```

# JavaFX - our first window (2)

```
import javafx.application.Application;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    public static void main(String[] args){
        System.out.println("We are starting ...");
        launch(args);
        System.out.println("Are we stopping?");
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("First GUI window!");
        primaryStage.show();
    }

}
```



# JavaFX - our first window (2)

```
import javafx.application.Application;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    public static void main(String[] args){
        System.out.println("We are starting ...");
        launch(args);
        System.out.println("Are we stopping?");
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("First GUI window!");
        primaryStage.show();
    }

}
```

Application class for UI windows

main launches start
(main can be omitted!!!)

A Stage (here called primaryStage) is created automatically by JavaFX and passed as an argument to start()
A Stage is a window
Show() makes it appear



# Application class

JavaFX programs include **one** class that extends Application (analogous to a single class with a main method for console programs).

javafx.application.Application



# Application class

When running an Application class (a class that extends it), JavaFX does the following:

1. Constructs an instance of that Application class
2. Calls an `init()` method for application initialization
   … don't construct a Stage or Scene in `init()`
3. Calls the `start (javafx.stage.Stage)` method
4. Waits for the application to finish: you call `Platform.exit()` or the last window has been closed.
3. Calls the `stop()` method to release resources.
   init() and stop() have default do-nothing implementations.

# JavaFX - our first window (cont'd)

So we have a window (Stage),
what are we going to put in it?




# JavaFX - a button window




```
package examples;

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

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

    @Override
    public void start(Stage primaryStage) throws Exception {

        Button okBtn = new Button("ok");

        primaryStage.setTitle("First GUI window!");
        primaryStage.show();


    }

}
```

where is the button?



# JavaFX - a button window


```
package examples;

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

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

    @Override
    public void start(Stage primaryStage) throws Exception {

        Button okBtn = new Button("ok");

        Scene scene = new Scene (okBtn, 150, 100);
        primaryStage.setScene(scene);

        primaryStage.setTitle("First GUI window!");
        primaryStage.show();
    }
}
```



# JavaFX - a button window


```
package examples;

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

public class HelloButton extends Application{
    pu
    }
    @O
    pu
```

1. Create UI control
2. Add to a scene (scene graph, all UI components)
3. Set the scene as the main scene of your stage (window), here called primaryStage
4. Show your stage to make the window visible

```
        Button okBtn = new Button("ok");
        Scene scene = new Scene (okBtn, 150, 100);
        primaryStage.setScene(scene);

        primaryStage.show();

    }

}
```

# Terminology

Stage

- represents windows, top level **container**
- many setter methods, e.g., `setTitle(), setWidth()`
- you can have multiple stages and use (**set**) one or the other as your main stage (primaryStage in our example):
  construct a Stage for each window in your application, e.g., for dialogs and pop-ups.

Scene

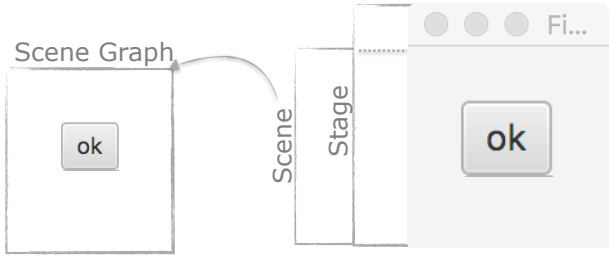
- each stage has a scene (scene graph container)
- scenes hold controls (Buttons, Labels, etc.)
- you can put controls directly in scenes, or use **Panes** for better layout hierarchies:
  construct scenes for collections of widgets you want to be grouped and visible together



# Basic structure

Basic structure of a JavaFX program

- Application
- Override the `start(Stage)` method
- Stage ← Scene ← Nodes (Panes or Controls)



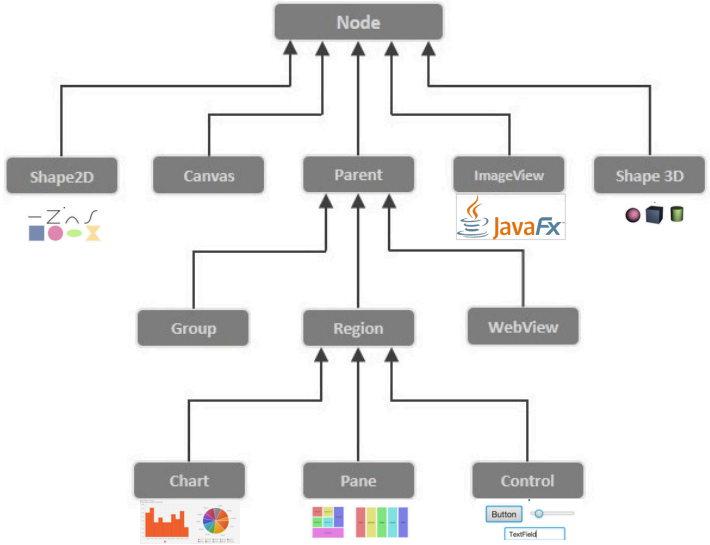



# widget complexity

- All UI elements are **Node**s in JavaFX
- Simple widgets (class Control)
  - buttons, scroll bars, labels, …
- Composite/complex widgets
  - containers that group other widgets (class Pane)
  - dialog boxes, menus, color pickers, …



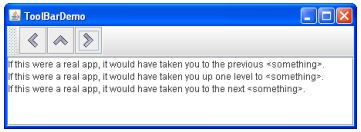
# UI hierarchy



# widget tree

Hierarchical representation of the widget structure

- a widget can belong to only one « container »

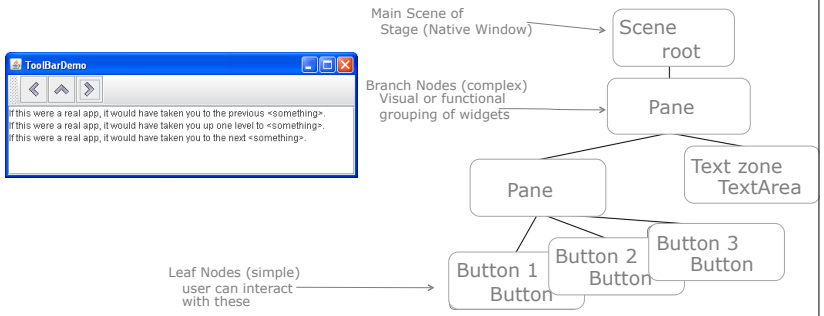





# widget tree

Hierarchical representation of the widget structure

- a widget can belong to only one « container »






# interface toolkits

All toolkits have:
a collection of UI controls / components
a way to layout these controls (next)
a way to handle input events from users

(syntax and command parameters may differ depending on language / toolkit)



# JavaFX layouts

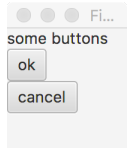
# JavaFX - complex structures

So we have a window (Stage), with a Button.
How can we add more controls on to it?




# JavaFX - a button window

```
package examples;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class SomeButtons extends Application {

    // removed main, Application takes care of starting the program
    // BUT only one per project

    @Override
    public void start(Stage primaryStage) throws Exception {

        Label descriptionLabel = new Label("some buttons");
        Button okBtn = new Button("ok");
        Button cancelBtn = new Button("cancel");

        VBox root = new VBox();
        root.getChildren().addAll(descriptionLabel, okBtn, cancelBtn);

        Scene scene = new Scene(root, 100, 100);

        primaryStage.setTitle("First GUI window!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

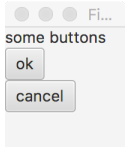





# JavaFX - a button window

```
package examples;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class SomeButtons extends Application {

    // removed main, Application takes care of starting the program
    // BUT only one per project

    @Override
    public void start(Stage primaryStage) throws Exception {

        Label descriptionLabel = new Label("some buttons");
        Button okBtn = new Button("ok");
        Button cancelBtn = new Button("cancel");

        VBox root = new VBox();
        root.getChildren().addAll(descriptionLa

        Scene scene = new Scene(root, 100, 100);

        primaryStage.setTitle("First GUI window!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

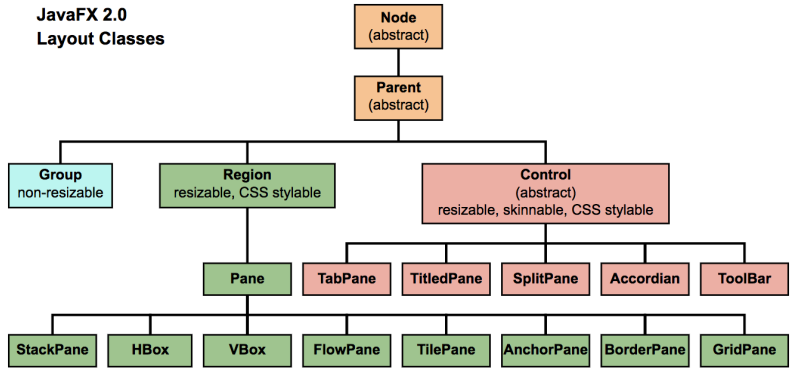
VBox will store nodes/controls and be added to a Scene






# JavaFX - complex structures

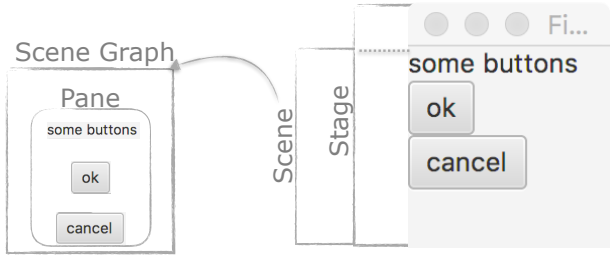
VBox is one of many *Pane* class objects
that help us organize nodes in a container



# A more realistic structure

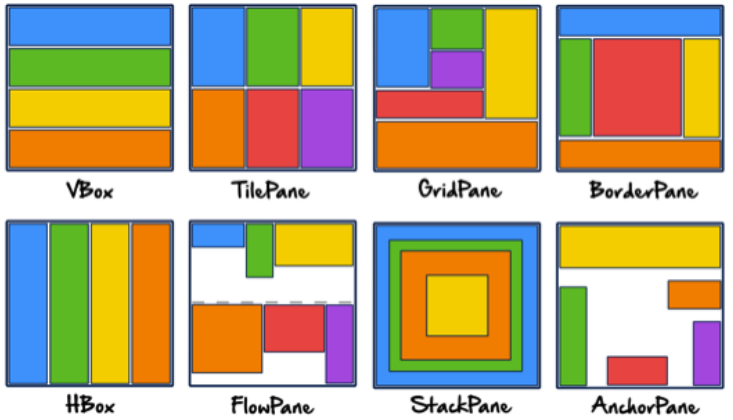
A more realistic structure of a JavaFX program

- Application
- Override the `start(Stage)` method
- Stage ← Scene ← Panes ← UI Nodes






# Panes for layout

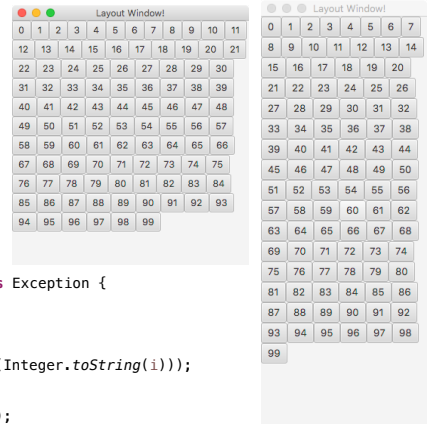



more tutorials at https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm
image from JavaFX 8 By Hendrik Ebbers & Michael Heinrichs



# Panes for layout - FlowPane

```
package examples;

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

public class LayoutButtons extends Application {

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

      @Override
      public void start(Stage primaryStage) throws Exception {

            FlowPane root = new FlowPane();

            for (int i = 0; i<100; ++i) {
                  root.getChildren().add(new Button(Integer.toString(i)));
            }

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

            primaryStage.setTitle("Layout Window!");
            primaryStage.setScene(scene);
            primaryStage.show();
      }
}
```




https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm



# Panes for layout - GridPane

```
public class LayoutGrid extends Application {

      @Override
      public void start(Stage primaryStage) throws Exception {

            GridPane root = new GridPane();
            root.setVgap(10);
            root.setHgap(10)
            int i =0;
            int j =0;

            for (i=0;i<10; ++i)
                  for (j=0; j<10;++j){
                        Button btn = new Button(i + "." + j);
                        root.add( btn, i,j); // not use getChildren() as we set position
            }


            Text txt = new Text ("This is a grid");
            root.add(txt,0,j,10,1); // start column,row ; span column,row

            Scene scene = new Scene(root, 500, 500);

            primaryStage.setTitle("Layout Window!");
            primaryStage.setScene(scene);
            primaryStage.show();
      }
}
```


https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

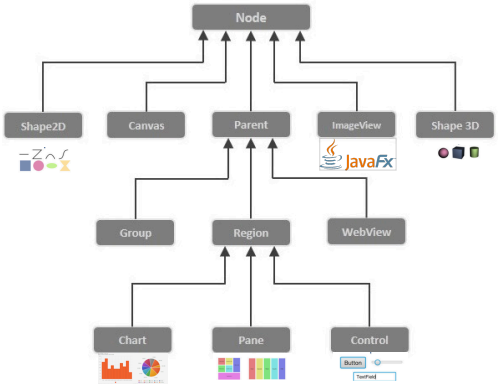
# Layout Pane Classes

| | |
|---|---|
| Pane | PaneBase class for all layout panes. It contains the getChildren() method that returns all **nodes** in the pane. |
| Stack Pane | Nodes on top of each other in center of pane. |
| FlowPane | Nodes flow to fill horizontal (vertical) space. |
| GridPane | Nodes in cell(s) of a grid. |
| BorderPane | Nodes in one of five regions (T,B,C,L,R). |
| AnchorPane | Nodes anchored on sides or center of pane. |
| HBox | Nodes horizontally. |
| VBox | Nodes vertically. |



# The class Node

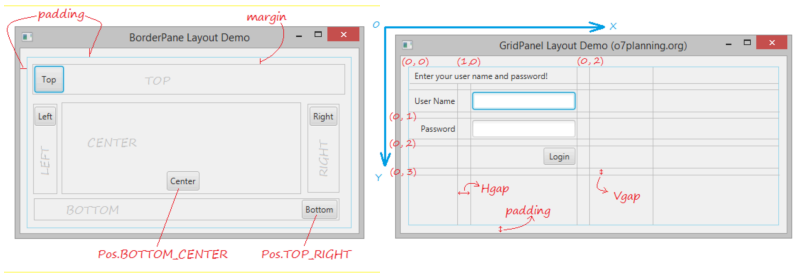
Layout Panes are considered nodes (same as Controls) so they can be added to other Panes






# Improving layout

Layout Panes have different properties to help create layouts that persist during resizing (margin, padding, Vgap/Hgap, alignment)




https://o7planning.org/en/10629/javafx-borderpane-layout-tutorial



# Panes for layout - examples

```
import javafx.application.Application;

public class CombinedLayouts extends Application{

    @Override
    public void start(Stage primaryStage) throws Exception {

        HBox hbox = new HBox();
        hbox.setPadding(new Insets(15, 12, 15, 12)); // padding all around
        hbox.setSpacing(10); // space between nodes
        hbox.setStyle("-fx-background-color: #336699;"); // familiar?

        Button buttonCurrent = new Button("Current");
        buttonCurrent.setPrefSize(100, 20); // preferred size

        Button buttonProjected = new Button("Projected");
        buttonProjected.setPrefSize(100, 20);
        hbox.getChildren().addAll(buttonCurrent, buttonProjected);

        BorderPane root = new BorderPane();
        root.setTop(hbox); // a Pane added to another Pane

        Scene scene = new Scene (root, 200, 200);
        primaryStage.setTitle("Complex Window!");
        primaryStage.setScene(scene);
        primaryStage.show();

    }
}
```

https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

# building the interface

When coding (after the design phase):

- Always start by laying out nodes in the window
- Handle the functionality (discussed next) **after**.

Use **Pane**s to **structure** and
**sub-divide** the layout.



# building the interface

Example of structure and resulting code:

```
Scene
Pane
    Label "A"
    TextField
Panel
    Label "B"
    TextField
```

**Structure**

```
Container panel = getContentPane();

VBox panelA = new VBox();
panel.add(panelA);
panelA.add(new Label("A"));
panelA.add(new TextField(5));

VBox panelB = new VBox();
panel.add(panelB);
panelB.add(new Label("B"));
panelB.add(new TextField(5));
```

**Code**



# JavaFX and CCS



# Consistent design

Imagine we have one or more windows and decide we want to change their visual style everywhere …

CSS (cascading style sheets)

… it describes how HTML elements are to be displayed on screen, paper, or in other media …

… and saves a lot of work. It can control the layout of multiple web pages all at once

# Consistent design

In the location of your main create a CSS file
New → Other → General → mycss.css

Inside the css add some styling properties:

```
.root {
     -fx-background-image: url("background.jpeg");
}

.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 );
}
.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 );
}

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

https://docs.oracle.com/javafx/2/get_started/css.htm



# Original Class

```
public class UseNoCSS extends Application {

     @Override
     public void start(Stage primaryStage) throws Exception {

          GridPane grid = new GridPane();
          grid.setAlignment(Pos.CENTER);
          grid.setHgap(10);
          grid.setVgap(10);
          grid.setPadding(new Insets(25, 25, 25, 25));

          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);

          Button okBtn = new Button("ok");
          grid.add(okBtn, 1,3);

          Scene scene = new Scene(grid, 300, 275);
          primaryStage.setScene(scene);
          primaryStage.setTitle("Trying without CSS window!");
          primaryStage.show();
     }
}
```

https://docs.oracle.com/javafx/2/get_started/css.htm



# Original Class + CSS commands

```
public class UseCSS extends Application {
     @Override
     public void start(Stage primaryStage) throws Exception {

          GridPane grid = new GridPane();
          grid.setAlignment(Pos.CENTER);
          grid.setHgap(10);
          grid.setVgap(10);
          grid.setPadding(new Insets(25, 25, 25, 25));

          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);

          Button okBtn = new Button("ok");
          grid.add(okBtn, 1,3);

          Scene scene = new Scene(grid, 300, 275);

          scene.getStylesheets().add
           (UseCSS.class.getResource("mycss.css").toExternalForm());

          primaryStage.setTitle("Trying with CSS");
          primaryStage.setScene(scene);
          primaryStage.show();
     }
}
```

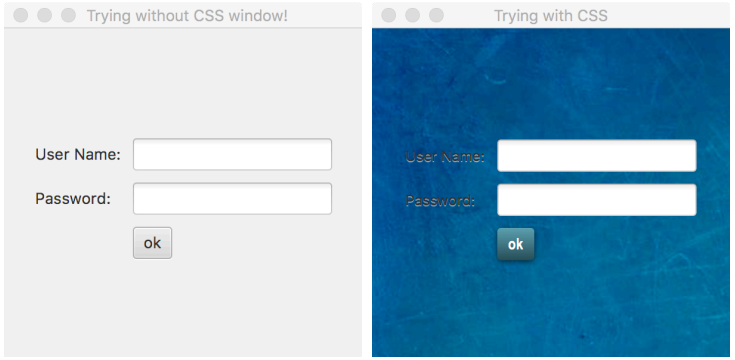
Simple CSS use to apply basic styling: just load CSS file

https://docs.oracle.com/javafx/2/get_started/css.htm



# Consistent design using CSS

Simple way to apply style to all windows




https://docs.oracle.com/javafx/2/get_started/css.htm

# Resources

https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

https://docs.oracle.com/javase/8/javafx/layout-tutorial/size_align.htm#JFXLY133

https://o7planning.org/en/11009/javafx (lots on layouts)

https://docs.oracle.com/javafx/2/get_started/css.htm



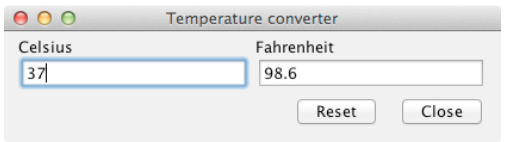
# In-class exercise



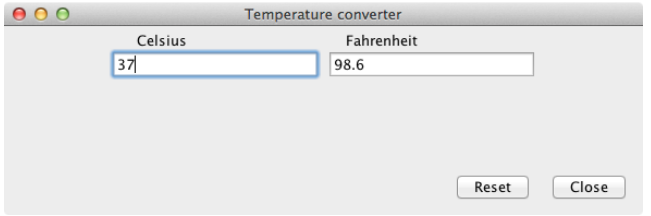
# In-class assignment

Go to our website:
(this will look a bit like your TA, but you'll focus on different layouts and behavior)

We will work on these together step by step (decide on layout)

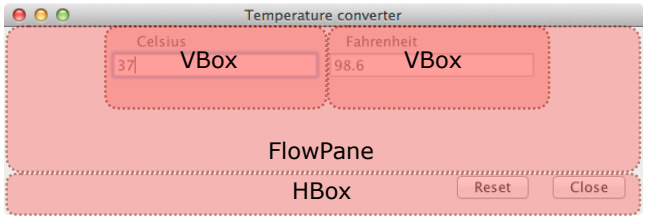





Before writing code for the layout, **identify the structure** that sub-divides nicely into rectangular areas. (*)

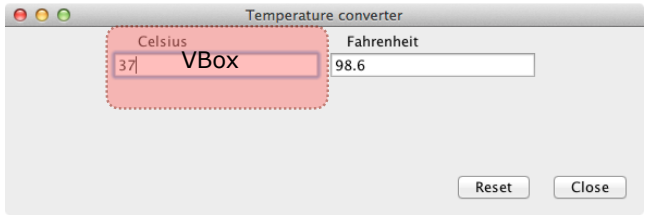
(*) in TA tomorrow you'll try another solution




In this example, we have sub-divided the layout into different Panes and chosen a specific layout (VBox and FlowPane) for each of them.

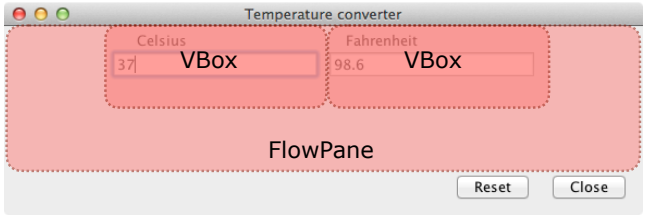
Note that there is not a unique solution.






```
VBox paneC = new VBox();
paneC.setPadding(new Insets(10, 10, 10, 10));
paneC.getChildren().addAll(new Label("Celcius"), new TextField());
```

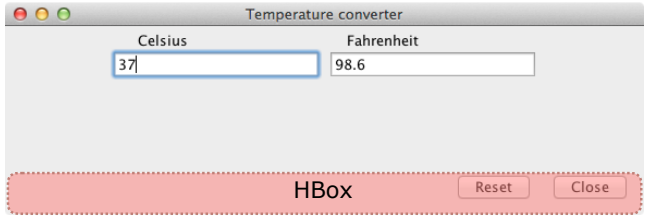




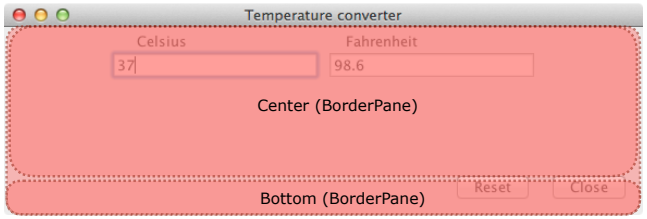

```
FlowPane paneCF = new FlowPane();
paneCF.getChildren().addAll(paneC,paneF);
```

Insertion order is important: items are added from **left to right** for horizontal layouts and **top to bottom** for vertical layouts.






```
HBox paneButtons = new HBox();
paneButtons.setPadding(new Insets(10, 10, 10, 10));
paneButtons.setSpacing(10);
paneButtons.setAlignment(Pos.CENTER_RIGHT);
paneButtons.getChildren().addAll(new Button ("Reset"),
                                 new Button("Close"));
```




```
BorderPane root = new BorderPane();
root.setCenter(paneCF);
root.setBottom(paneButtons);
```



# In-class assignment

Consider your minesweeper,
how would you layout your controls?