Introduction to Programming

W11 Graphical User Interfaces

Stephan Krusche

7 January 2026
Technical University of Munich

Schedule

#	Date	Subject
1	15.10.25	Introduction
2	22.10.25	Control Flow in Programming
3	29.10.25	**In-Depth Core Concepts***
4	05.11.25	Core Data Structures
5	12.11.25	Code Reuse and Structure
6	19.11.25	Type Flexibility and Safety
7	26.11.25	**In-Depth Object Orientation***
8	03.12.25	Functional Programming Essentials
9	10.12.25	Algorithms and Data Handling
10	17.12.25	Programming Languages
11	**07.01.26**	**Graphical User Interfaces**
12	14.01.26	Recursion
13	21.01.26	Concurrency
14	28.01.26	Beyond Programming
15	04.02.26	Course Review

*** Repetition**

Roadmap

- **Context**
  - Apply OOP concepts: abstraction, encapsulation, inheritance and polymorphism
  - Use control structures, data types, enums, annotations, generics, collections, iterators, lambda expressions, and streams
  - Apply error handling, implement algorithms, and understand the concept of programming languages
- **Learning goals**
  - Understand the importance of usability and prototyping
  - Differentiate between different graphical user interface frameworks
  - Explain the concept of model view controller
  - Implement layouts, shapes and controls in JavaFX
  - Style controls and shapes in JavaFX

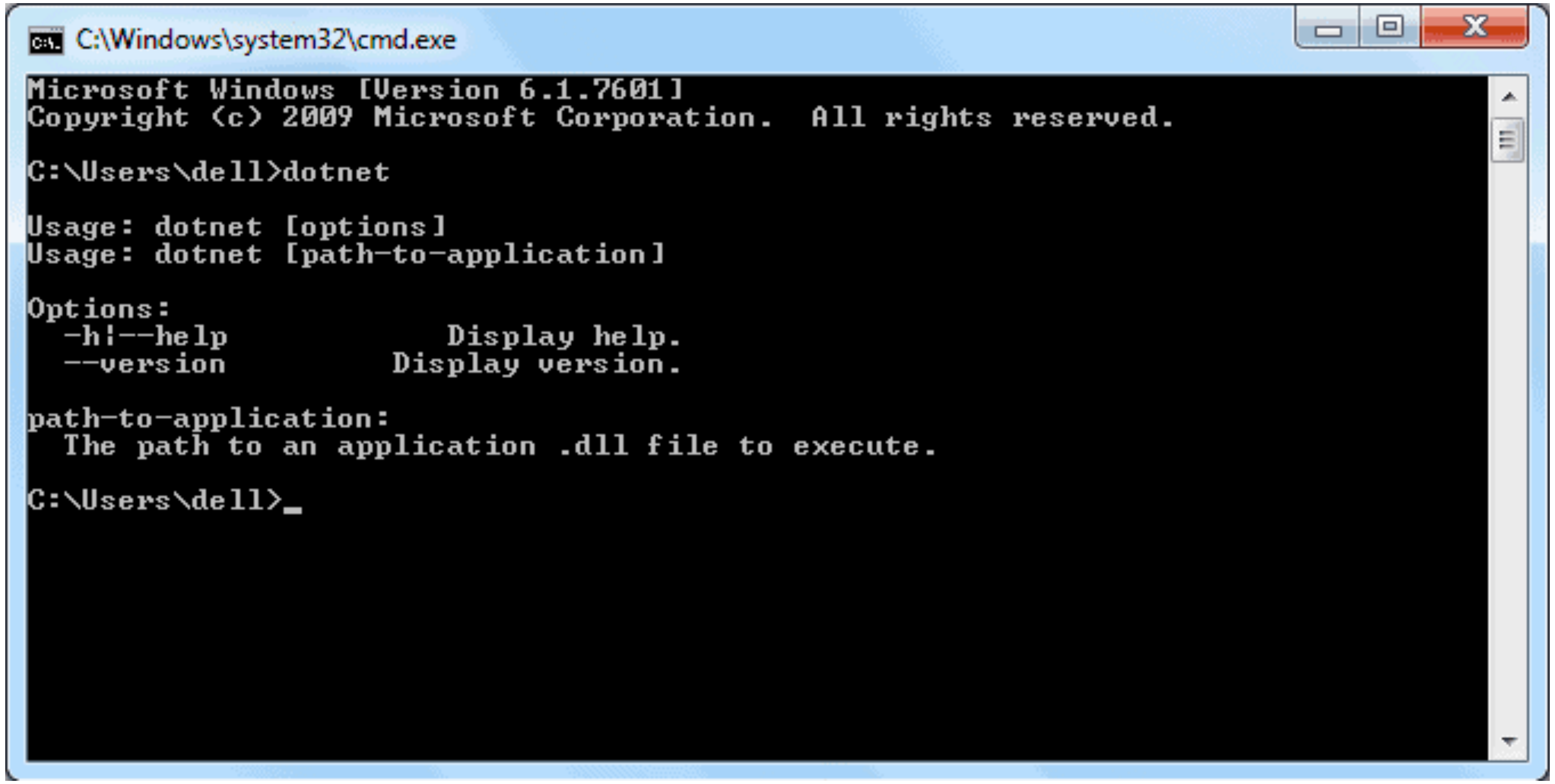
Outline

- ➡ Usability
- JavaFX
- Layout
- User input
- Shapes
- Styling

Graphical user interface (GUI)

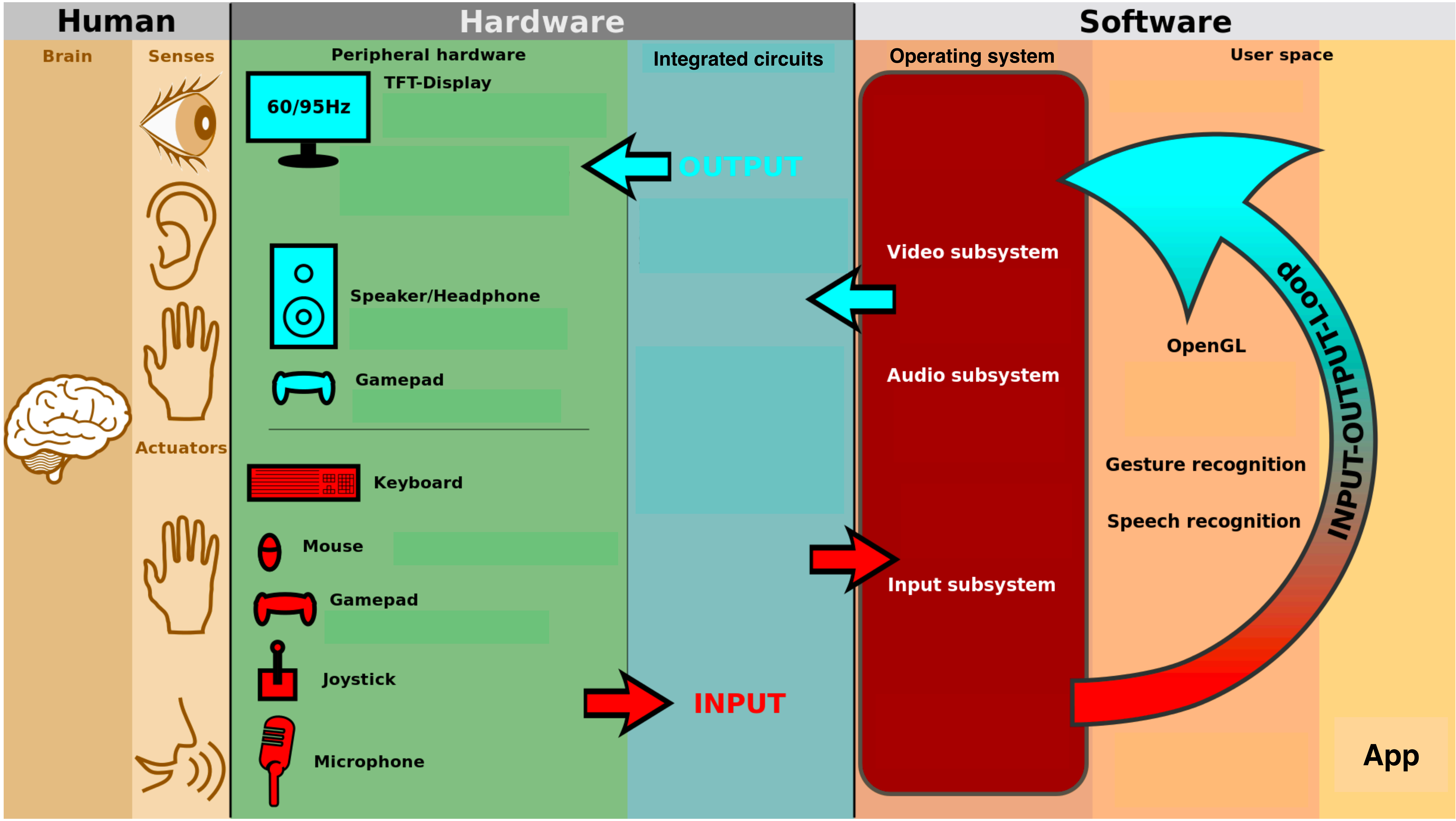
- Enables a person (user) to communicate with a computer through the use of symbols, visual metaphors, and pointing devices
- Provides **user-friendly interaction**
- Introduced in reaction to the perceived steep learning curve of command-line interfaces (CLI)
- Actions in a GUI are usually performed through direct manipulation of the graphical elements

Command line interface / terminal

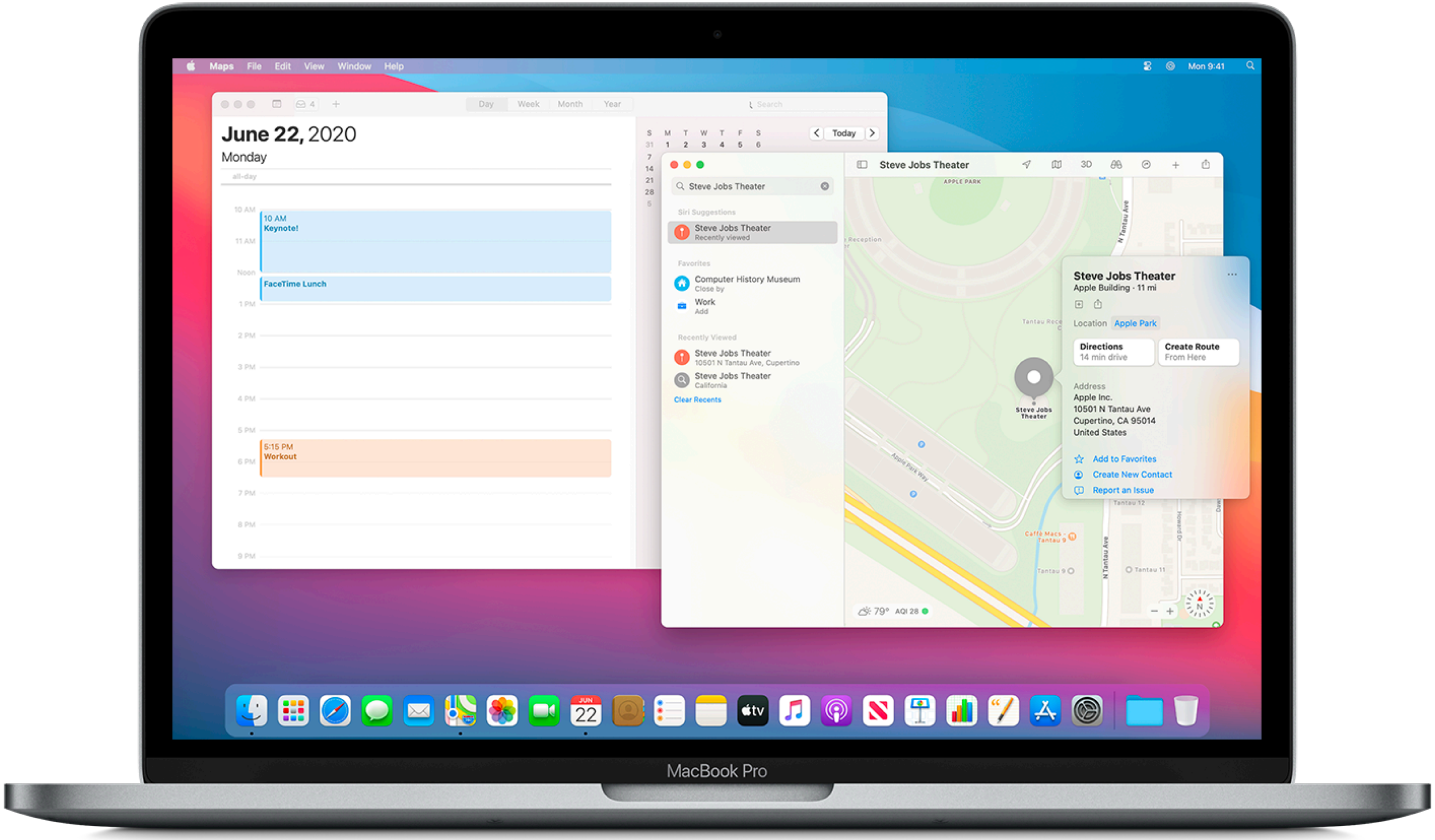


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\dell>dotnet

Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help            Display help.
  --version         Display version.

path-to-application:
  The path to an application .dll file to execute.

C:\Users\dell>_
```

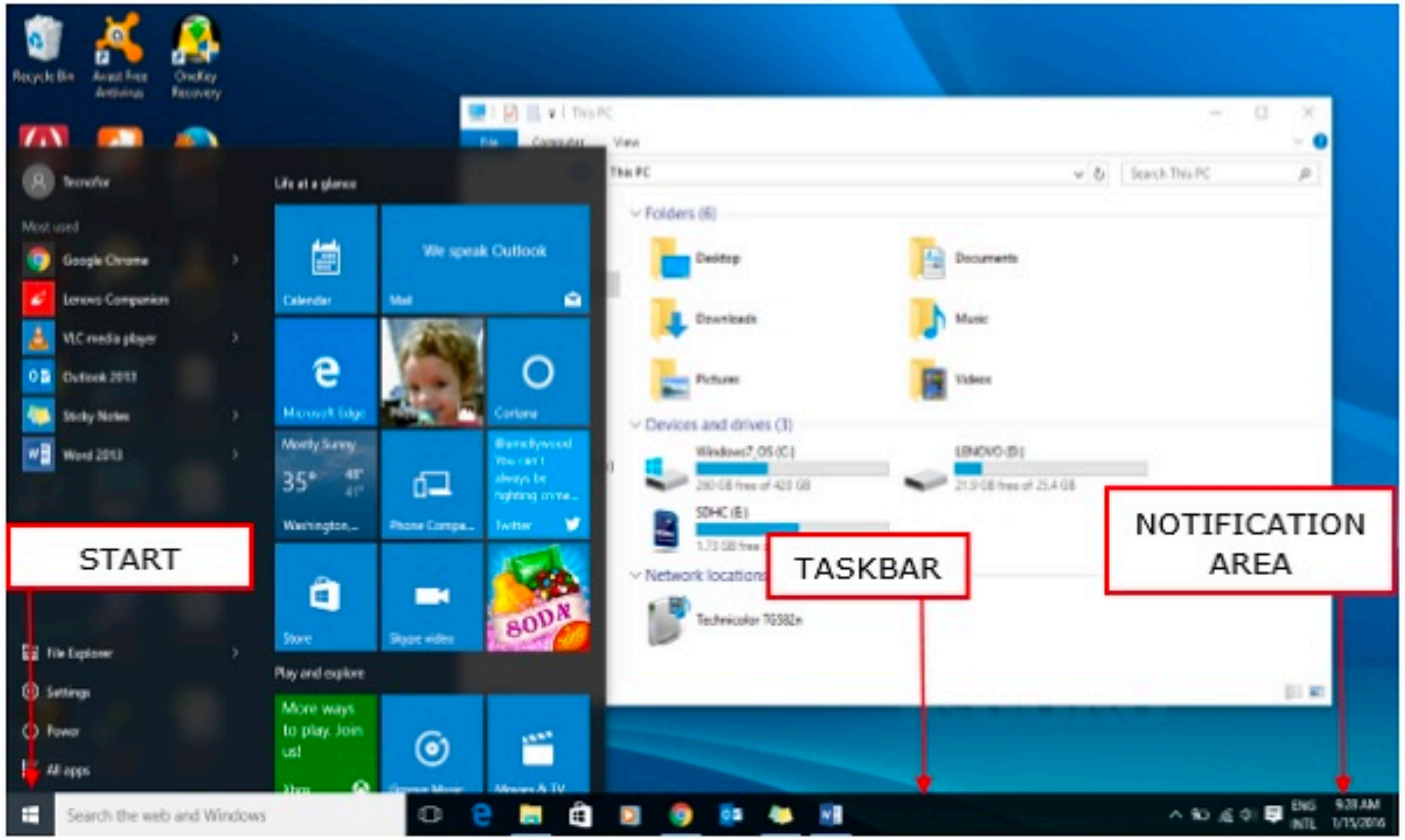
User interface and interaction design



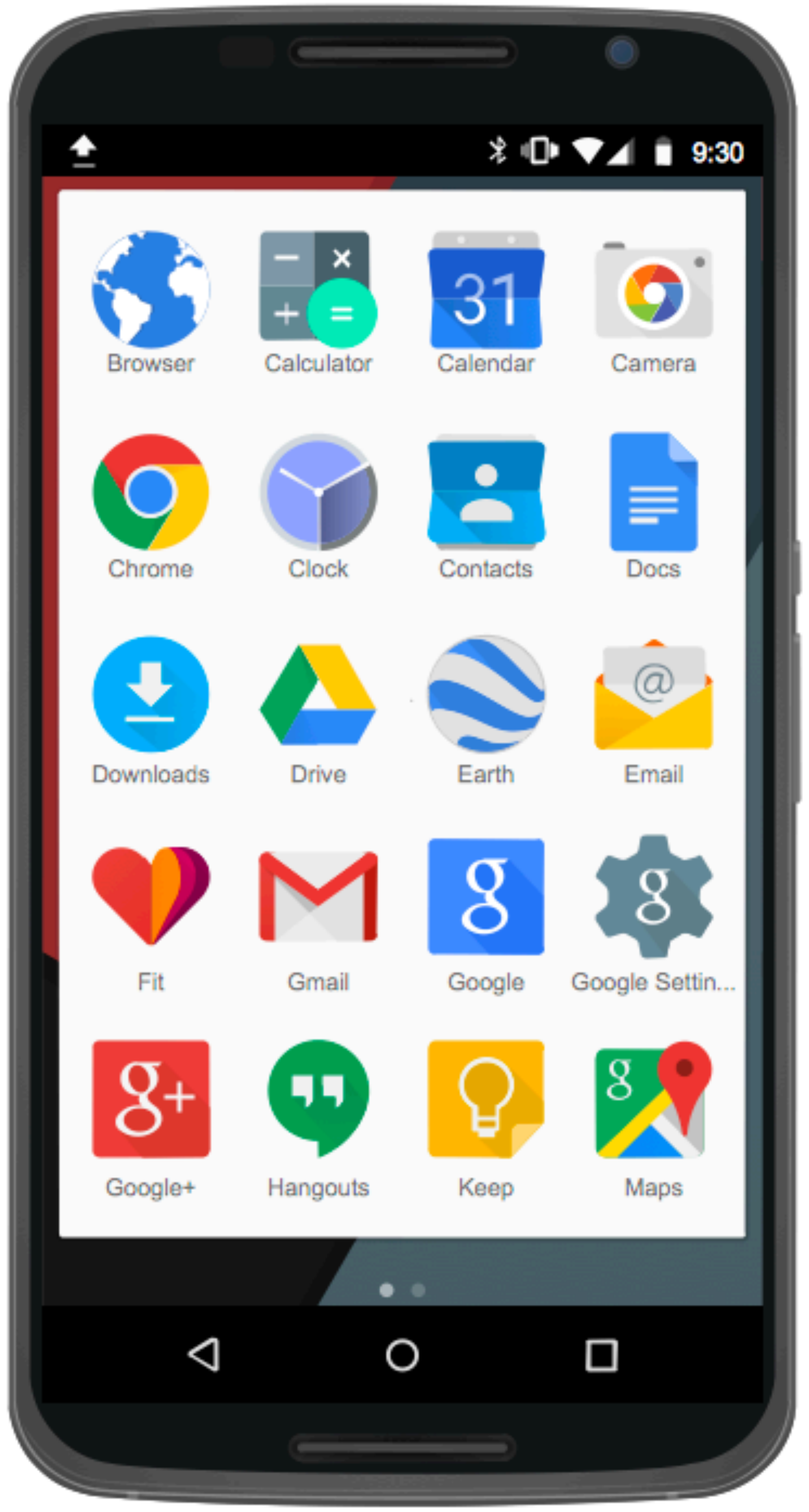
Example: macOS



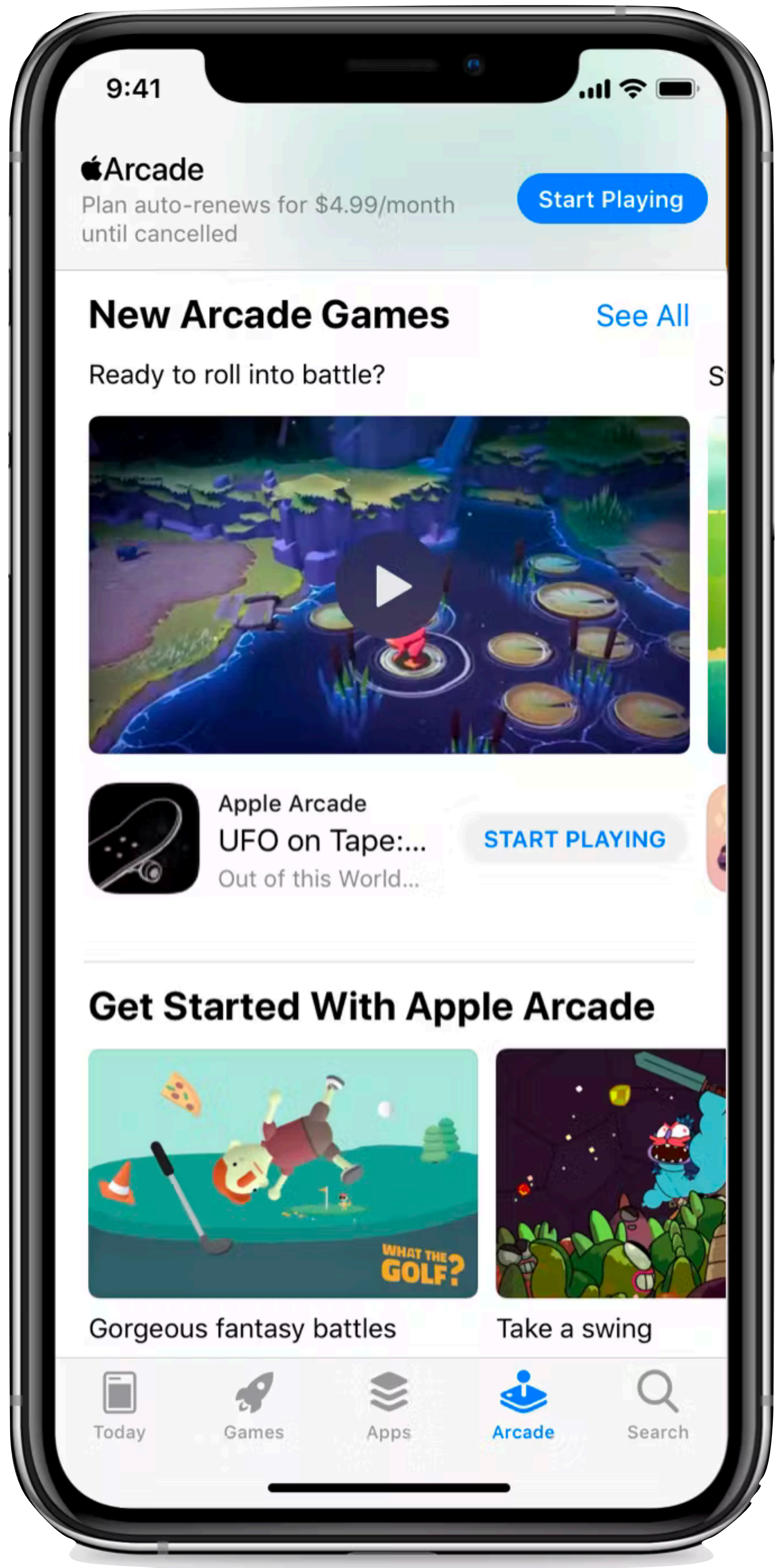
Example: Windows



Example: Android



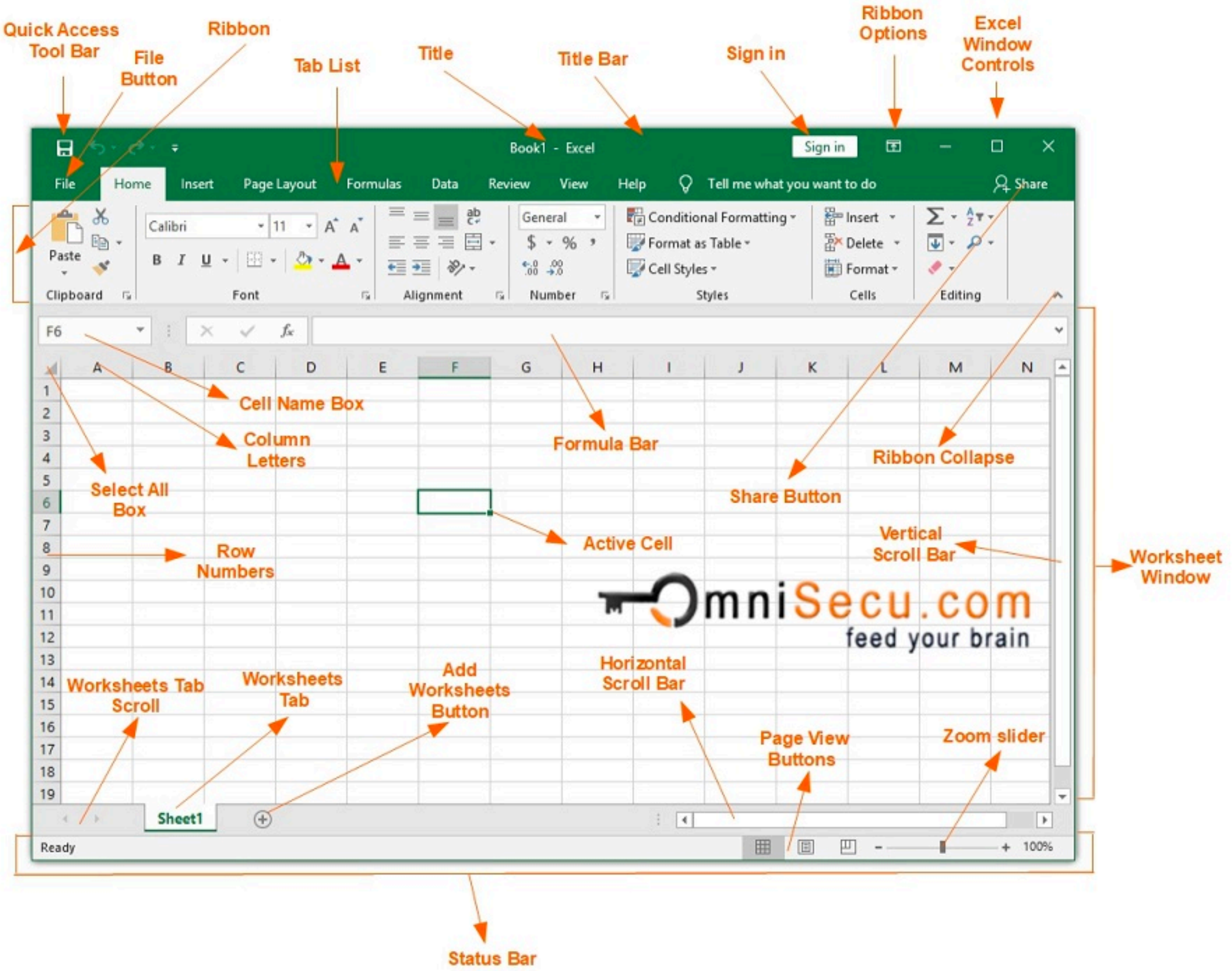
Example: iOS



Plan auto-renews for $4.99/month until cancelled

Ready to roll into battle?

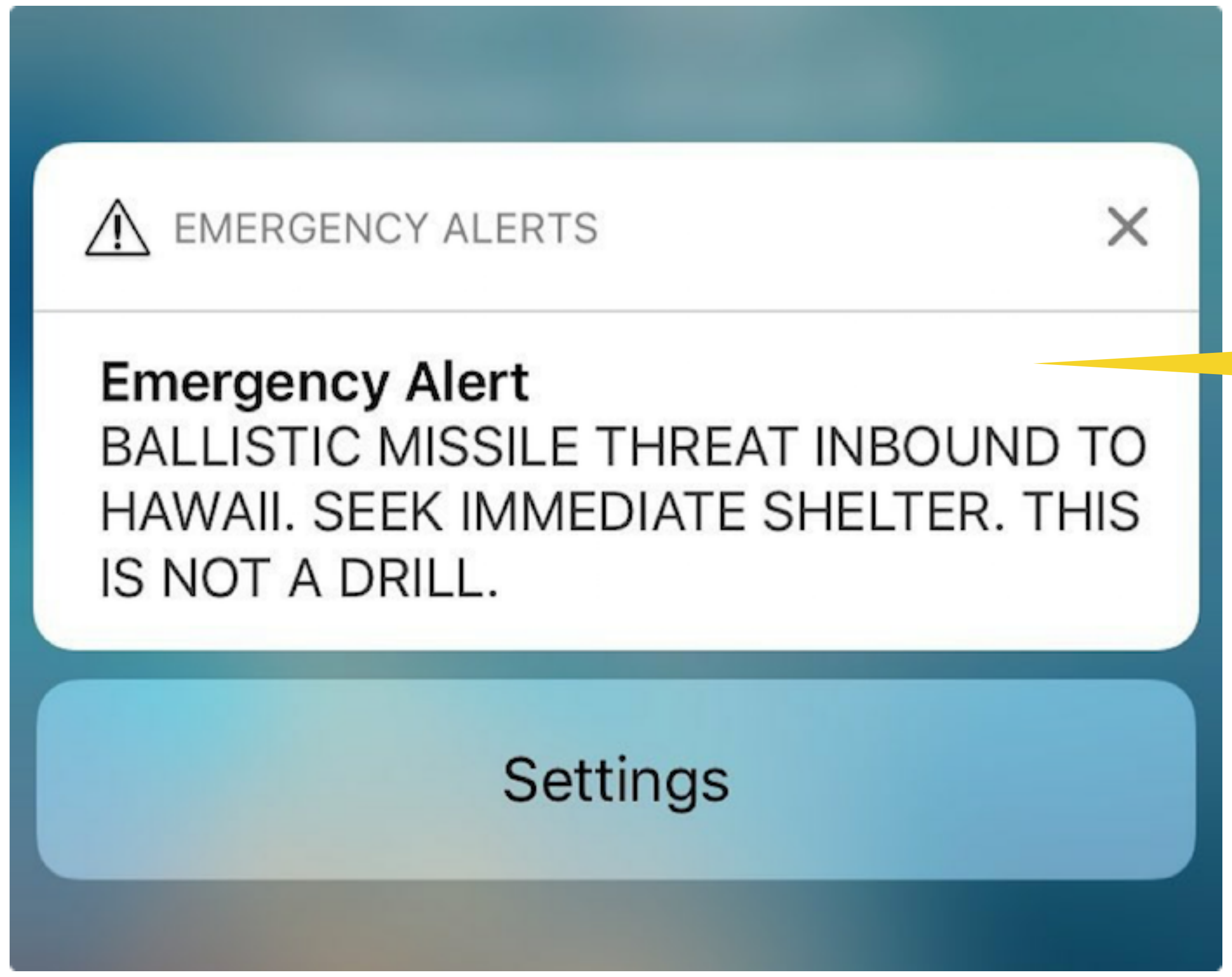
Example: Excel



Example: control rooms in nuclear power plants

Example: false Hawaiian missile alert

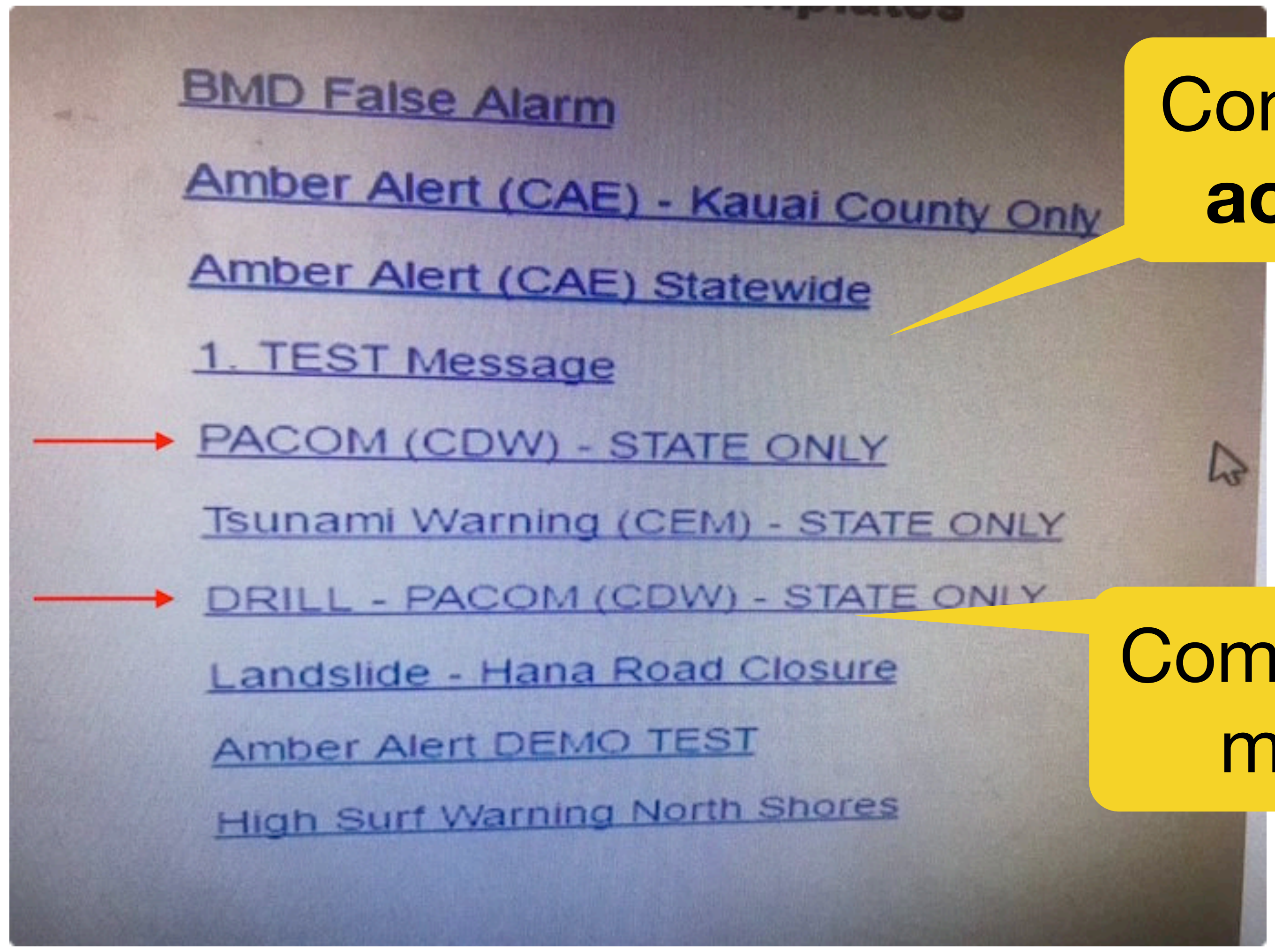
Example of a user interface disaster (see K. Flaherty)



EMERGENCY ALERTS

Emergency Alert
BALLISTIC MISSILE THREAT INBOUND TO HAWAII. SEEK IMMEDIATE SHELTER. THIS IS NOT A DRILL.

Message received by residents of Hawaii



BMD False Alarm
Amber Alert (CAE) - Kauai County Only
Amber Alert (CAE) Statewide
1. TEST Message
PACOM (CDW) - STATE ONLY
Tsunami Warning (CEM) - STATE ONLY
DRILL - PACOM (CDW) - STATE ONLY
Landslide - Hana Road Closure
Amber Alert DEMO TEST
High Surf Warning North Shores

Command to send an **actual** missile alert

Command to **test** missile alert

What happened?

- On January 13, 2018, an emergency alert was sent to the residents of Hawaii to warn them of the danger
- Fortunately, this was a false alarm!

What is the problem with the user interface?

- Poorly differentiated options
- Possibly no confirmation screen
  → Developers should not underestimate users' stress
- Problematic presentation or interaction design
  → Designer and user model gap

Usability

Measures how well a user can utilize the system functionality based on five categories

1. **Learnability**: how easily/fast can a user learn to use the system?
2. **Efficiency**: how many steps does it take a user to complete a particular task?
3. **Memorability**: how quickly can a user reestablish proficiency?
4. **Errors**: how many errors do users make, how severe are these errors, and how easily can they recover from the errors?
5. **Satisfaction (user experience)**: how pleasant is the design of the user interface?

Usability

- “The system is easy to use” — one of the most frequently misused terms, especially in advertising (often these systems are actually unusable)
- “Unusability” — the user has extreme difficulty in learning how to use or in using the system

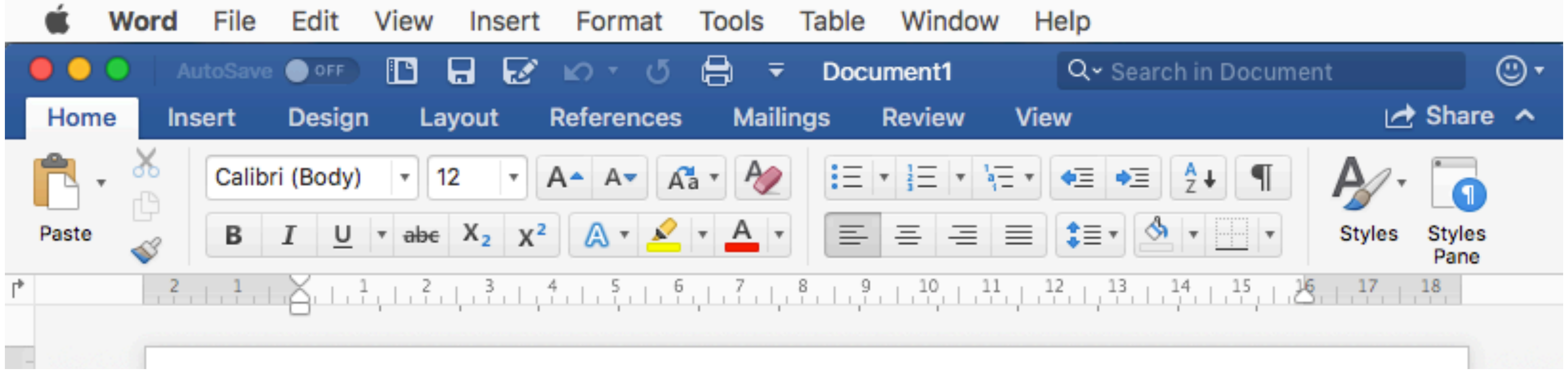
- Jakob Nielsen (2009): Anybody can do usability https://www.nngroup.com/articles/anybody-can-do-usability
- “*Usability is like cooking: everybody needs the results, anybody can do it reasonably well with a bit of training, and yet it takes a master to produce a gourmet outcome*”

User interfaces are hard to design

- The **developer** and the **user** are not the same person
  - Software engineers communicate mostly with other developers
  - User interface development is about communicating with users
- The **user is always right** …
  - Consistent problems are the system's fault
- … but the **user is not always right**
  - Users are no design experts
- User interface takes a lot of software development effort
  - ~50% of design, implementation and maintenance
- Managers must be involved (usability management)

Usability tradeoffs - example: learnability vs. efficiency

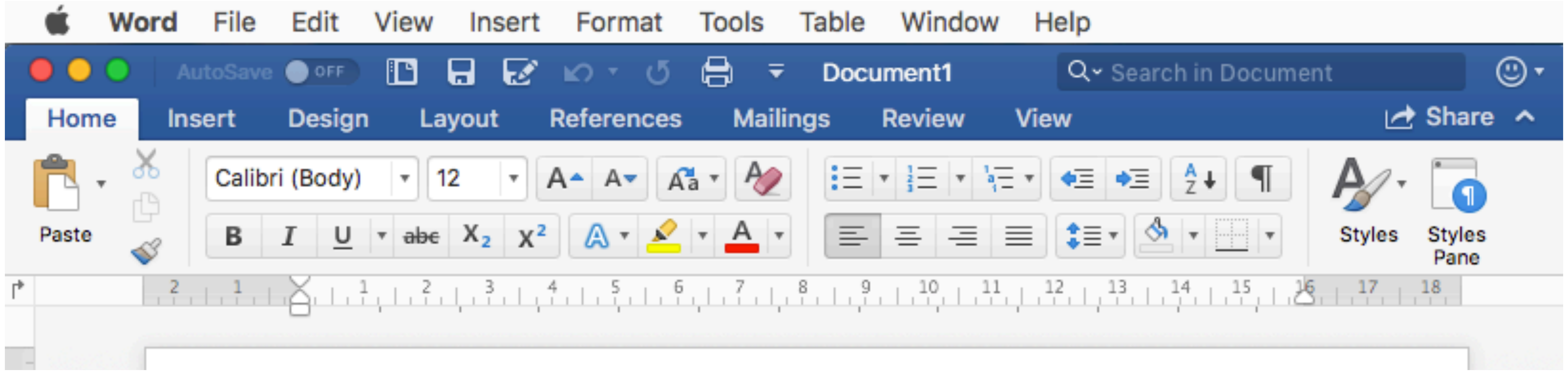
Question: how do you insert a table of contents into Microsoft Word?



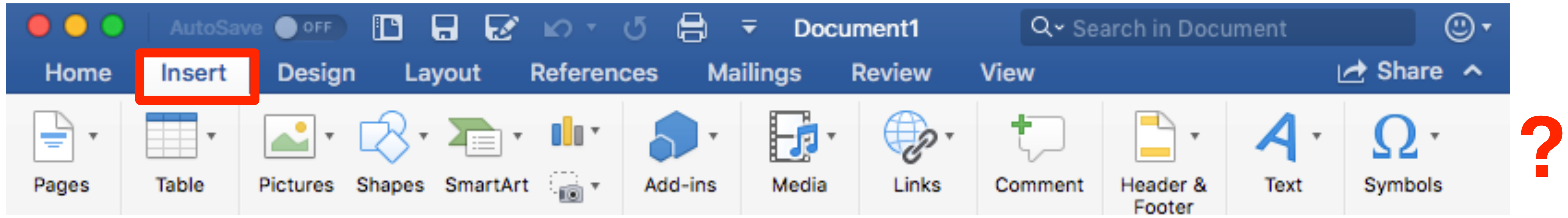
Word File Edit View Insert Format Tools Table Window Help
AutoSave OFF
Document1
Search in Document
Home Insert Design Layout References Mailings Review View
Share
Paste
Calibri (Body)
12
Styles
Styles Pane

Usability tradeoffs - example: learnability vs. efficiency

Question: how do you insert a table of contents into Microsoft Word?

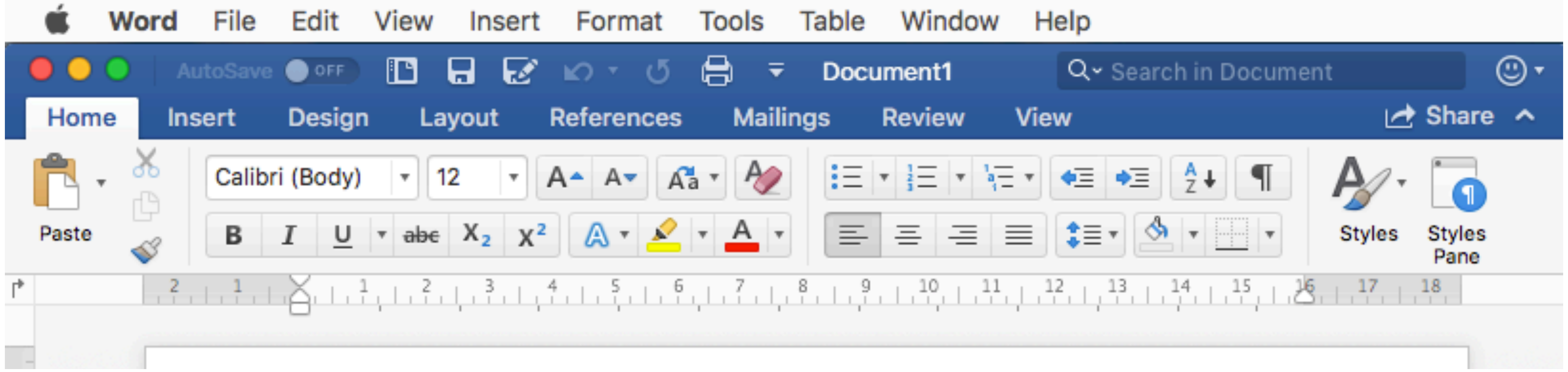

Word File Edit View Insert Format Tools Table Window Help
AutoSave OFF
Document1
Search in Document
Home Insert Design Layout References Mailings Review View
Share
Paste
Calibri (Body)
12
Styles
Styles Pane

1st try: click on “Insert” in the ribbon interface

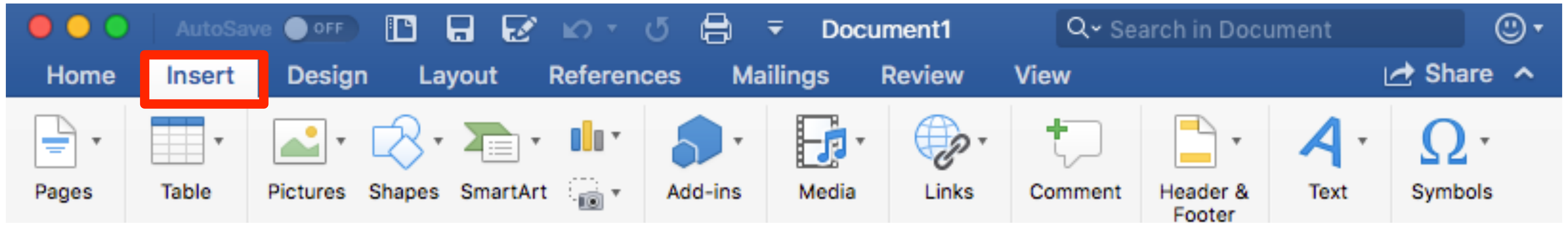


Usability tradeoffs - example: learnability vs. efficiency

Question: how do you insert a table of contents into Microsoft Word?

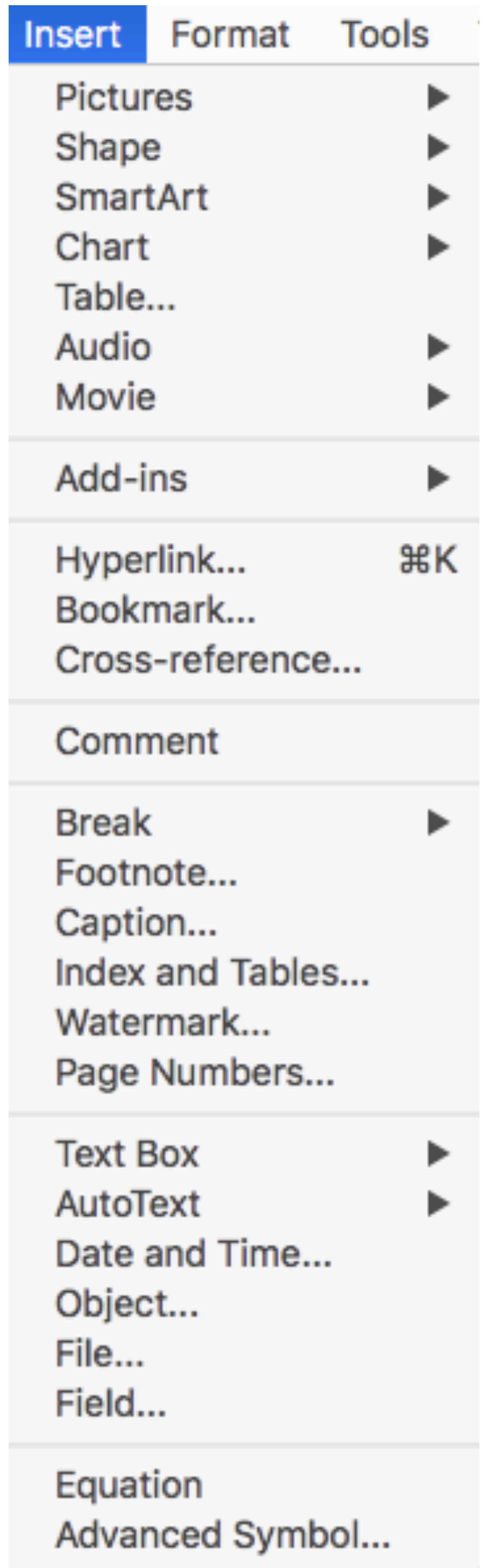


1st try: click on "Insert" in the ribbon interface



?

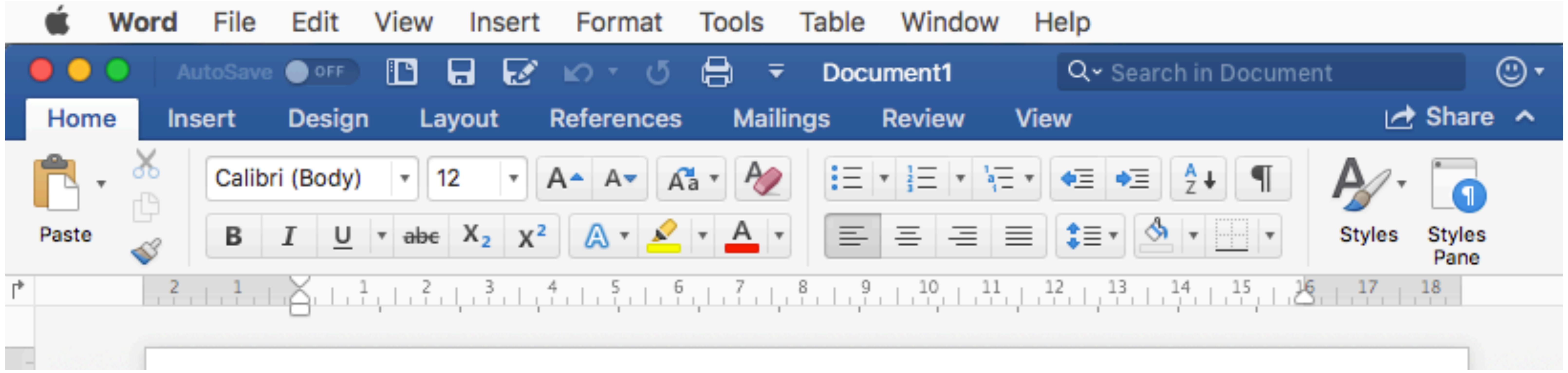
2nd try: click on "Insert" in the Menu



?

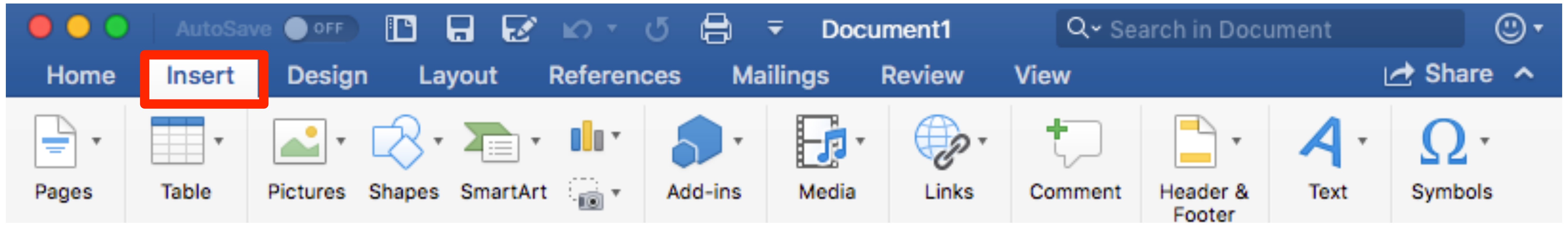
Usability tradeoffs - example: learnability vs. efficiency

Question: how do you insert a table of contents into Microsoft Word?



Word File Edit View Insert Format Tools Table Window Help
AutoSave OFF
Document1
Search in Document
Home Insert Design Layout References Mailings Review View
Share
Paste
Calibri (Body)
12
Styles
Styles Pane

1st try: click on “Insert” in the ribbon interface



?

Solution 1: click on “References” in the ribbon interface

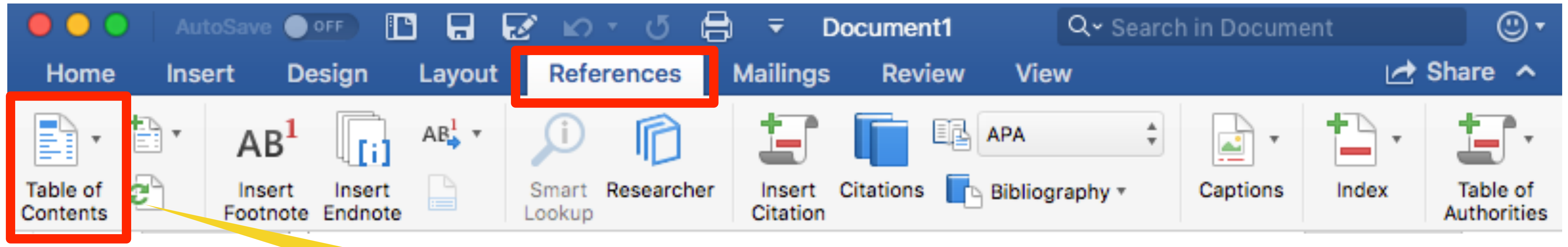
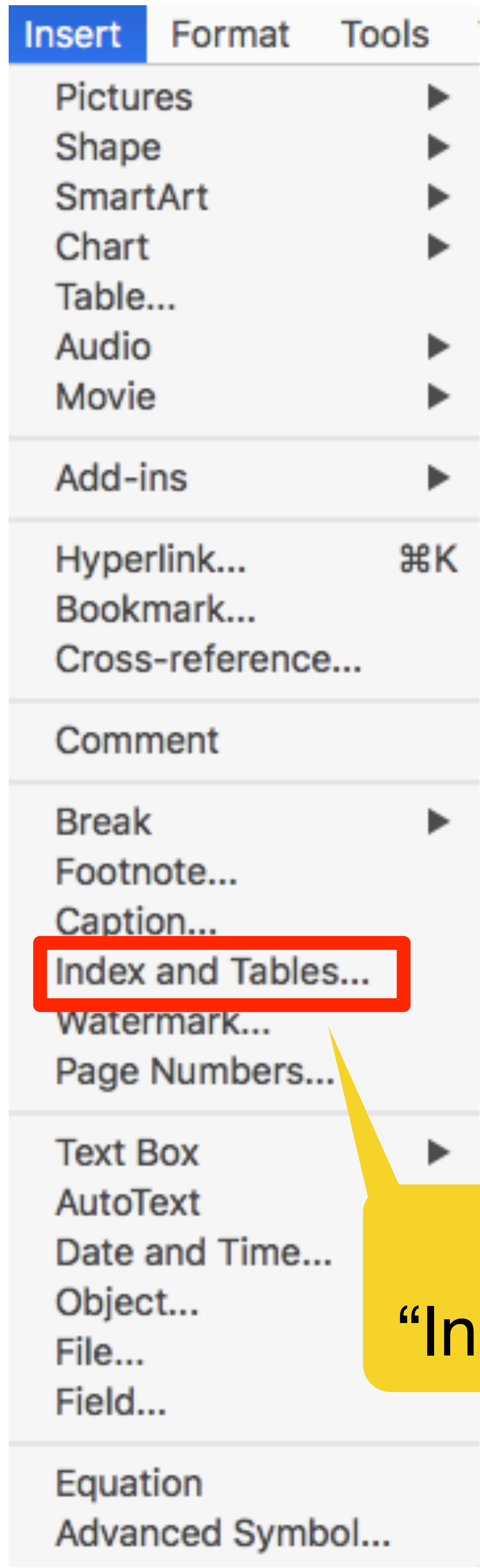


Table of Contents

2nd try: click on “Insert” in the Menu



?

Solution 2:
“Index and Tables…”

Prototyping

- “Prototyping is **externalizing** and making concrete a design idea for the purpose of evaluation.“ (Bill Verplank in Muñoz & Miller-Jacobs, 1992, S. 579)

- “Prototypes are for **traversing a design space**, leading to the creation of meaningful knowledge about the final design [...], and are purposefully formed manifestations of design ideas.“ (Lim et al., 2008, S. 3)

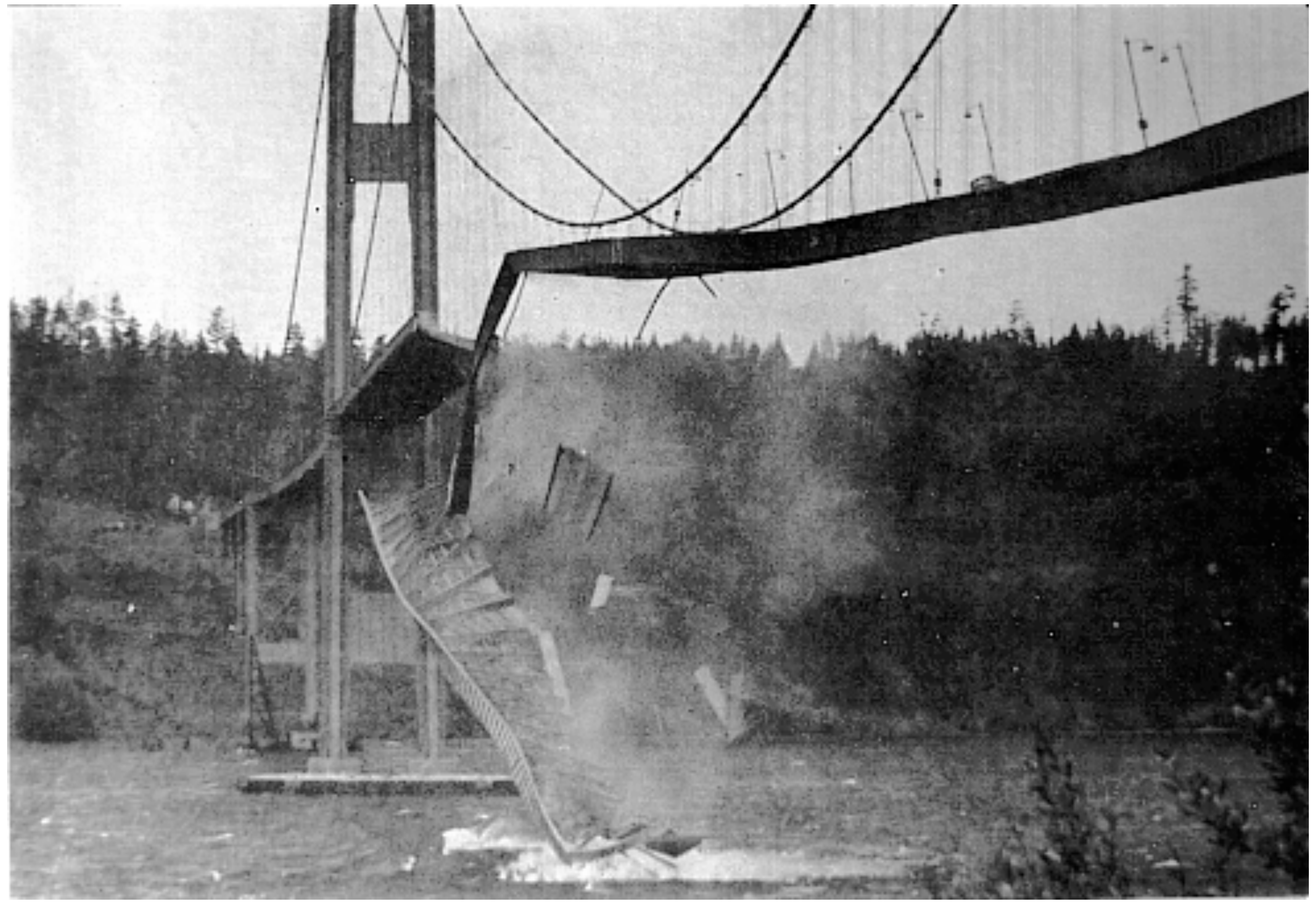
- “A prototype is an **early sample** or model built to test a concept or process or to act as a thing to be replicated or learned from.” (UXL Encyclopedia of Science)

Why prototyping?

- Instant **gratification**
- **Tangibility**: a prototype helps to understand a system early on
- **Improves communication**
- Allows **early decision-making**
- Mistakes can be found early
- “We want **instant prototypes**: they allow us to make more mistakes faster” (Elaine Hunt, Clemson University)
- Fast changes (flexibility) and small overhead

Failures are helpful

- Henry Petroski’s paradoxical approach to design
  - Better information comes from designs that fail rather than from those that succeed
  - Reason: failures draw more scrutiny
  - Petroski says without failure, complacency sets in
- Famous quote from Petroski: “Success in engineering is defined by its failures“
- “Destructive innovation”



Knowledge must be falsifiable

- Karl Popper (“objective knowledge”)
  - There is no absolute truth when trying to understand reality
  - One can only build theories that are “true” until somebody finds a counter example
- The truth of a theory is never certain
  - You can only use phrases like: “by our best judgment”, “using state of the art knowledge”
- Falsification: the act of disproving a theory or hypothesis

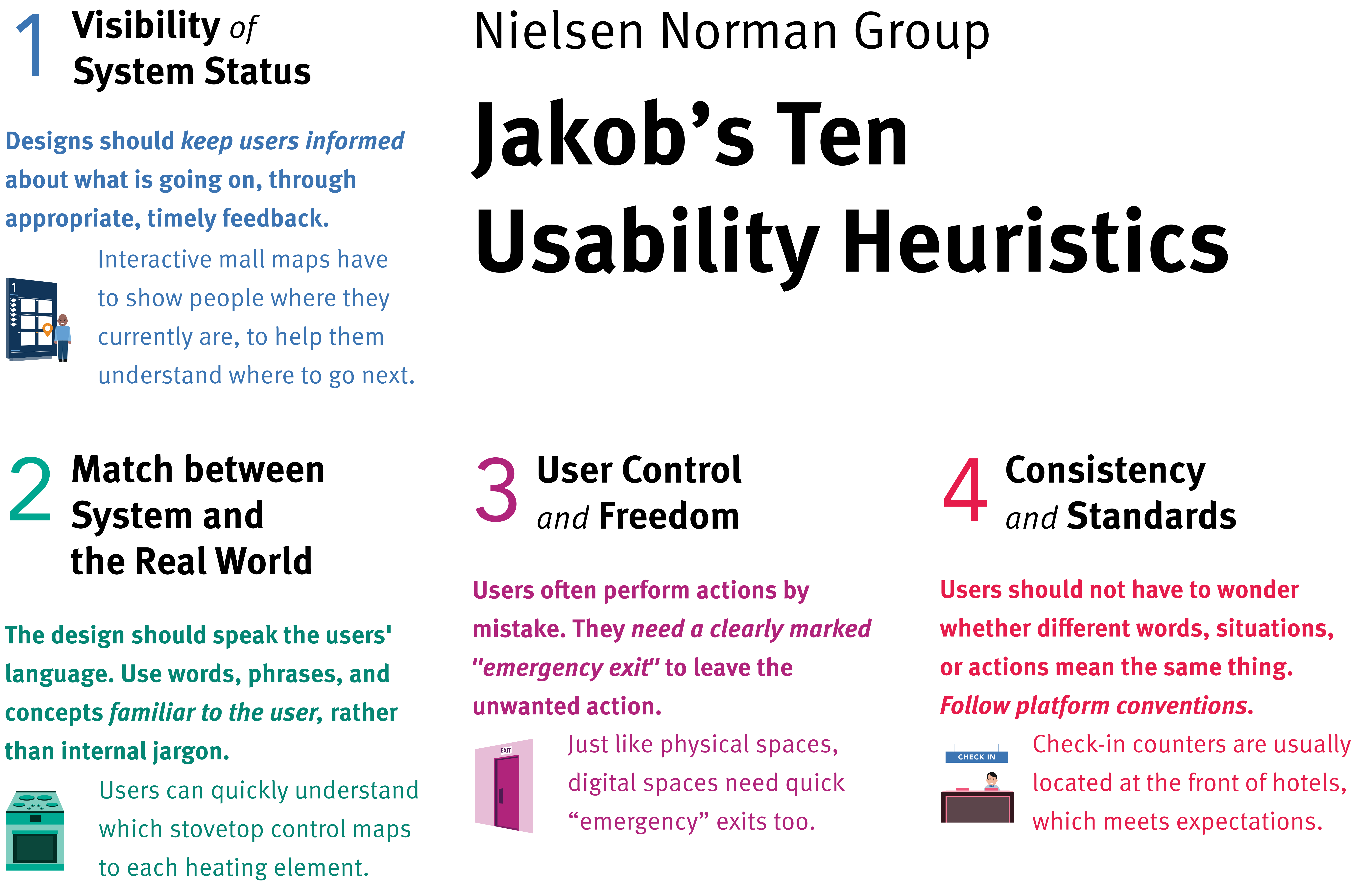
Consequence for software systems

- In software engineering, any system, including a user interface, is a model and thus a theory
  - We build models to find counterexamples
  - Techniques: requirements validation, user interface testing, review of the design, source code testing, system testing, etc
- Testing: the attempt of disproving a model

Methods to reach good usability

- Usability testing: watching a user interact with the system's user interface
    - Usability testing uses scenario-based design
    - Involves the creation of a test scenario
    - The user performs a list of tasks while the observer watches, takes notes, and compares the observed with the specified/expected behavior
- Heuristic evaluation: a usability engineering method to find usability problems in a user interface design

Nielsen’s 10 heuristics



Nielsen Norman Group

Jakob’s Ten Usability Heuristics

1 Visibility *of* System Status

Designs should *keep users informed* about what is going on, through appropriate, timely feedback.

Interactive mall maps have to show people where they currently are, to help them understand where to go next.

2 Match between System and the Real World

The design should speak the users' language. Use words, phrases, and concepts *familiar to the user,* rather than internal jargon.

Users can quickly understand which stovetop control maps to each heating element.

3 User Control *and* Freedom

Users often perform actions by mistake. They *need a clearly marked "emergency exit"* to leave the unwanted action.

Just like physical spaces, digital spaces need quick “emergency” exits too.

4 Consistency *and* Standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. *Follow platform conventions.*

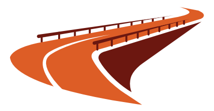
Check-in counters are usually located at the front of hotels, which meets expectations.

https://media.nngroup.com/media/articles/attachments/Heuristic_Summary1-compressed.pdf

Nielsen’s 10 heuristics (continued)

5 Error Prevention

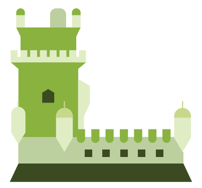
Good error messages are important, but the best designs carefully *prevent problems* from occurring in the first place.



Guard rails on curvy mountain roads prevent drivers from falling off cliffs.

6 Recognition Rather Than Recall

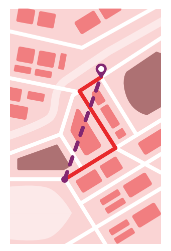
***Minimize the user's memory load* by making elements, actions, and options visible. Avoid making users remember information.**



People are likely to correctly answer “Is Lisbon the capital of Portugal?”.

7 Flexibility *and* Efficiency of Use

Shortcuts — hidden from novice users — may *speed up the interaction* for the expert user.



Regular routes are listed on maps, but locals with more knowledge of the area can take shortcuts.

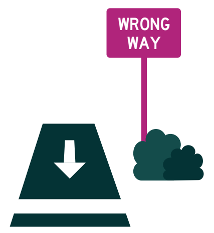
8 Aesthetic *and* Minimalist Design

Interfaces should not contain information which is irrelevant. Every extra unit of information in an interface *competes* with the relevant units of information.

A minimalist three-legged stool is still a place to sit.

9 Recognize, Diagnose, *and* Recover from Errors

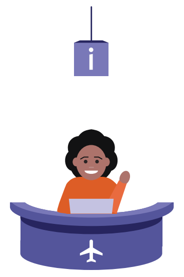
Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.



Wrong-way signs on the road remind drivers that they are heading in the wrong direction.

10 Help *and* Documentation

It’s best if the design *doesn’t need* any additional explanation. However, it may be necessary to provide documentation to help users complete their tasks.



Information kiosks at airports are easily recognizable and solve customers' problems in context and immediately.

https://media.nngroup.com/media/articles/attachments/Heuristic_Summary1-compressed.pdf

GUI frameworks

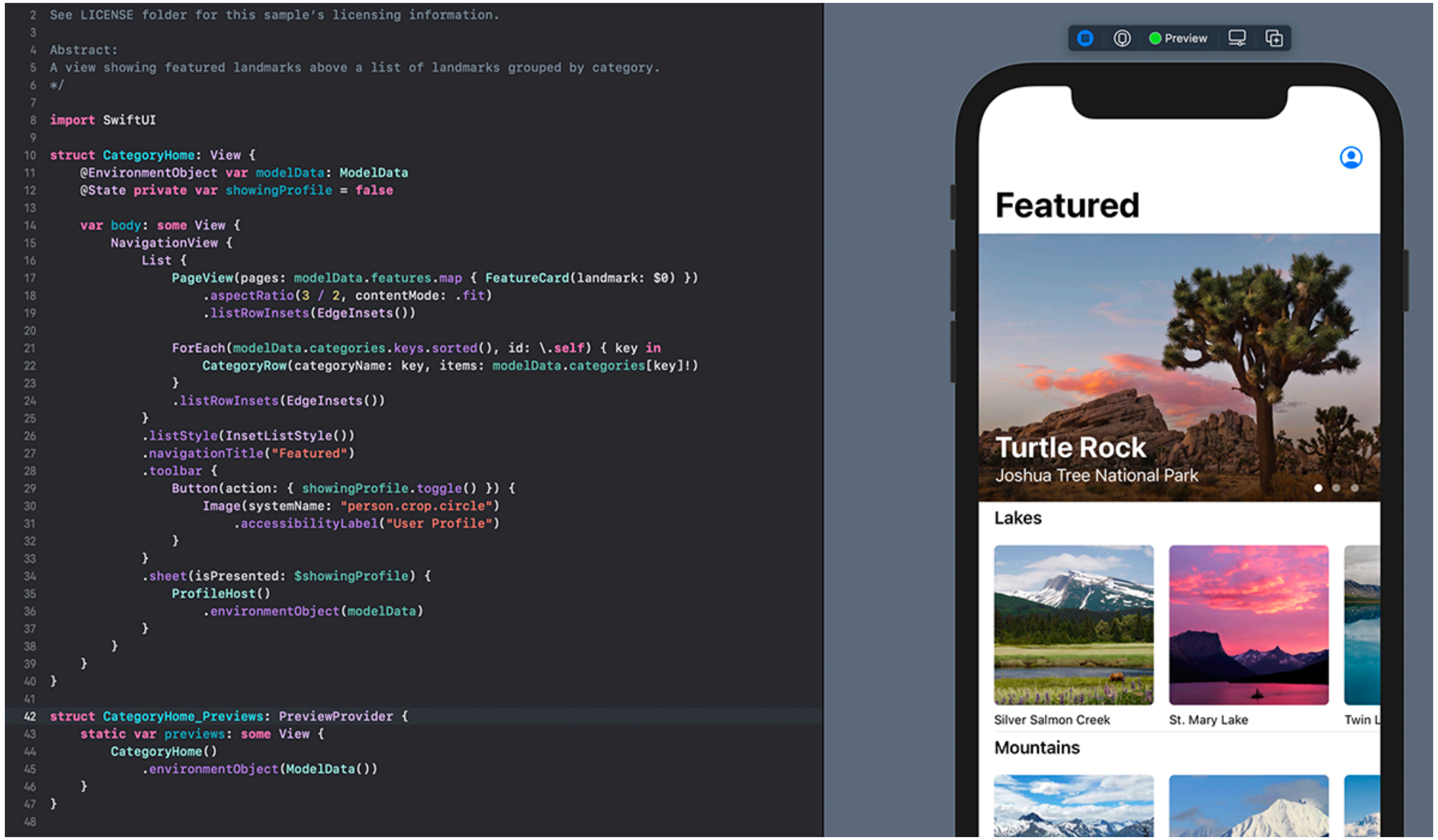
- **Web**: HTML and CSS
- **macOS / iOS**: Cocoa and Cocoa Touch, SwiftUI
- **.NET**: WinForms
- **Android**: Jetpack Compose
- **Java**: AWT, Swing, JavaFX

HTML

- Building block of the web: https://developer.mozilla.org/en-US/docs/Web/HTML
- Defines the meaning and structure of web content
- Companion technologies
  - Web page's appearance (CSS)
  - Web page's functionality (JavaScript)
- Provides basic user interface elements and layouts
  - Text, link, button, label, select, input, table
  - https://developer.mozilla.org/en-US/docs/Web/HTML/Element
- CSS allows defining style for these elements
  - Color, size, font, padding, margin, etc.
  - https://developer.mozilla.org/en-US/docs/Web/CSS

SwiftUI

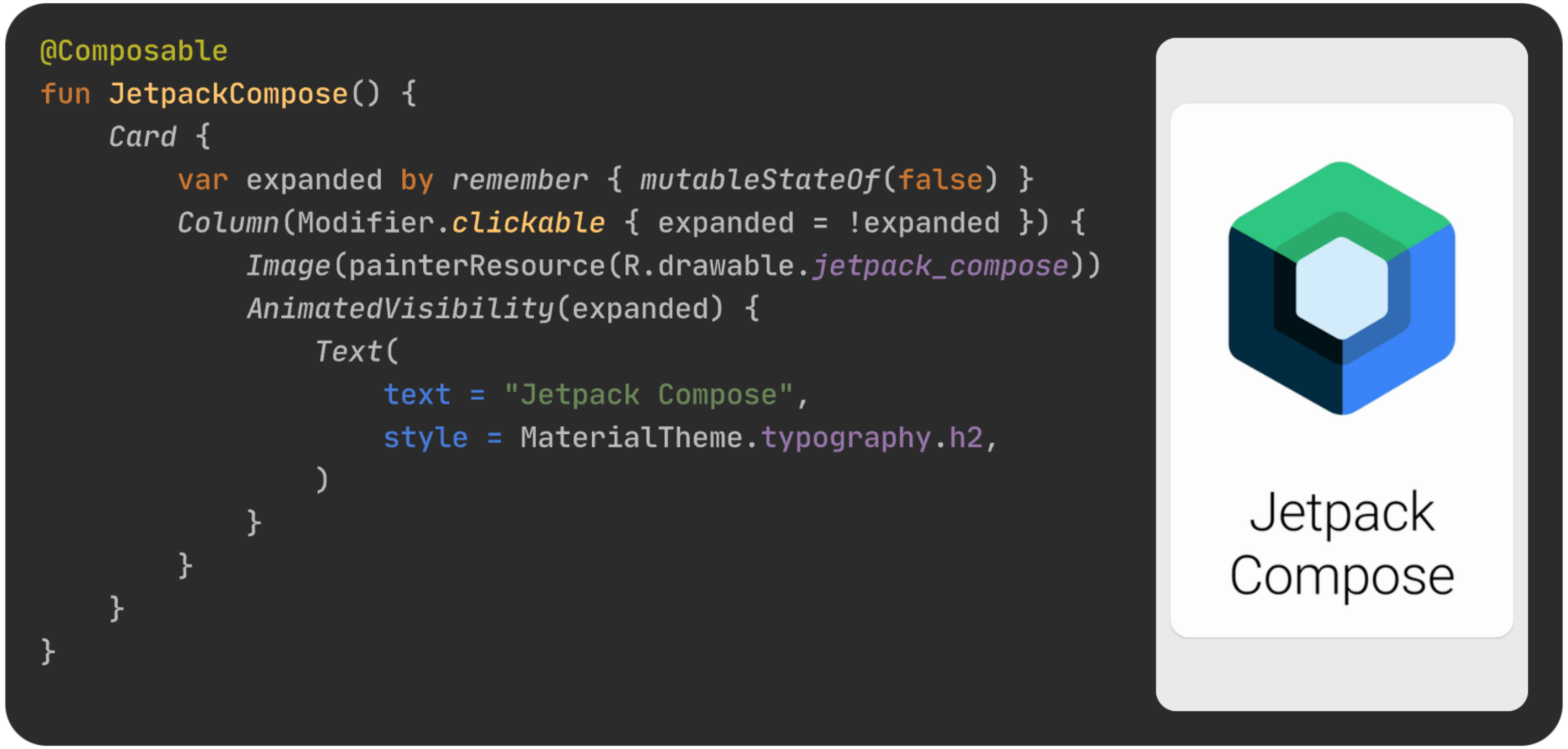
- Modern way to **declare** user interfaces for any Apple platform
- https://developer.apple.com/tutorials/swiftui



```
See LICENSE folder for this sample's licensing information.

Abstract:
A view showing featured landmarks above a list of landmarks grouped by category.
*/

import SwiftUI

struct CategoryHome: View {
    @EnvironmentObject var modelData: ModelData
    @State private var showingProfile = false

    var body: some View {
        NavigationView {
            List {
                PageView(pages: modelData.features.map { FeatureCard(landmark: $0) })
                    .aspectRatio(3 / 2, contentMode: .fit)
                    .listRowInsets(EdgeInsets())

                ForEach(modelData.categories.keys.sorted(), id: \.self) { key in
                    CategoryRow(categoryName: key, items: modelData.categories[key]!)
                }
                .listRowInsets(EdgeInsets())
            }
            .listStyle(InsetListStyle())
            .navigationTitle("Featured")
            .toolbar {
                Button(action: { showingProfile.toggle() }) {
                    Image(systemName: "person.crop.circle")
                        .accessibilityLabel("User Profile")
                }
            }
            .sheet(isPresented: $showingProfile) {
                ProfileHost()
                    .environmentObject(modelData)
            }
        }
    }
}

struct CategoryHome_Previews: PreviewProvider {
    static var previews: some View {
        CategoryHome()
            .environmentObject(ModelData())
    }
}
```

Jetpack Compose

- Modern toolkit to **declare** native user interfaces on Android
- https://developer.android.com/jetpack/compose



```kotlin
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.h2,
                )
            }
        }
    }
}
```

Outline

- Usability
- ➡ JavaFX
- Layout
- User input
- Shapes
- Styling

JavaFX

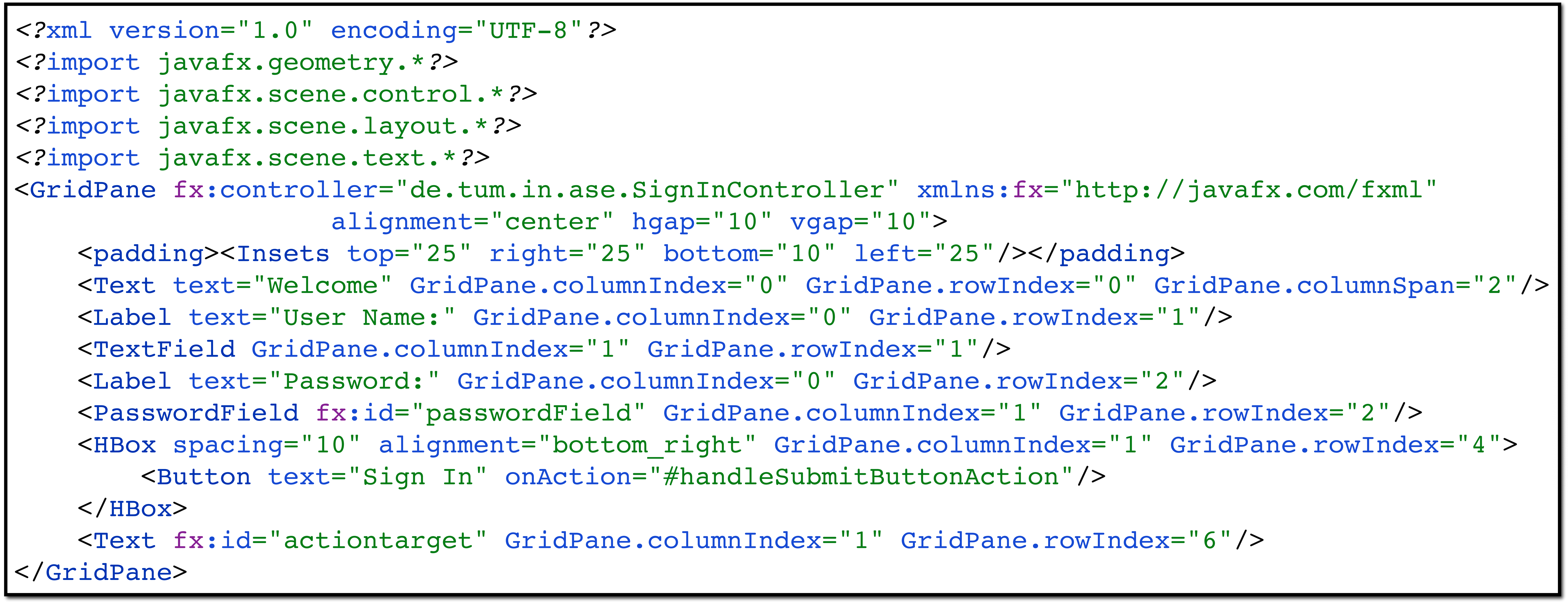
- Open source, next-generation client application platform for desktop, mobile and embedded systems built on Java: https://openjfx.io
- Great tutorial: https://jenkov.com/tutorials/javafx/index.html
- Comes with a large set of built-in GUI components, like buttons, text fields, tables, trees, menus, charts and much more
- Can be styled via CSS and/or programmatically
- Has support for 2D and 3D Graphics
- Has a **WebView** which can display modern web applications

JavaFX features

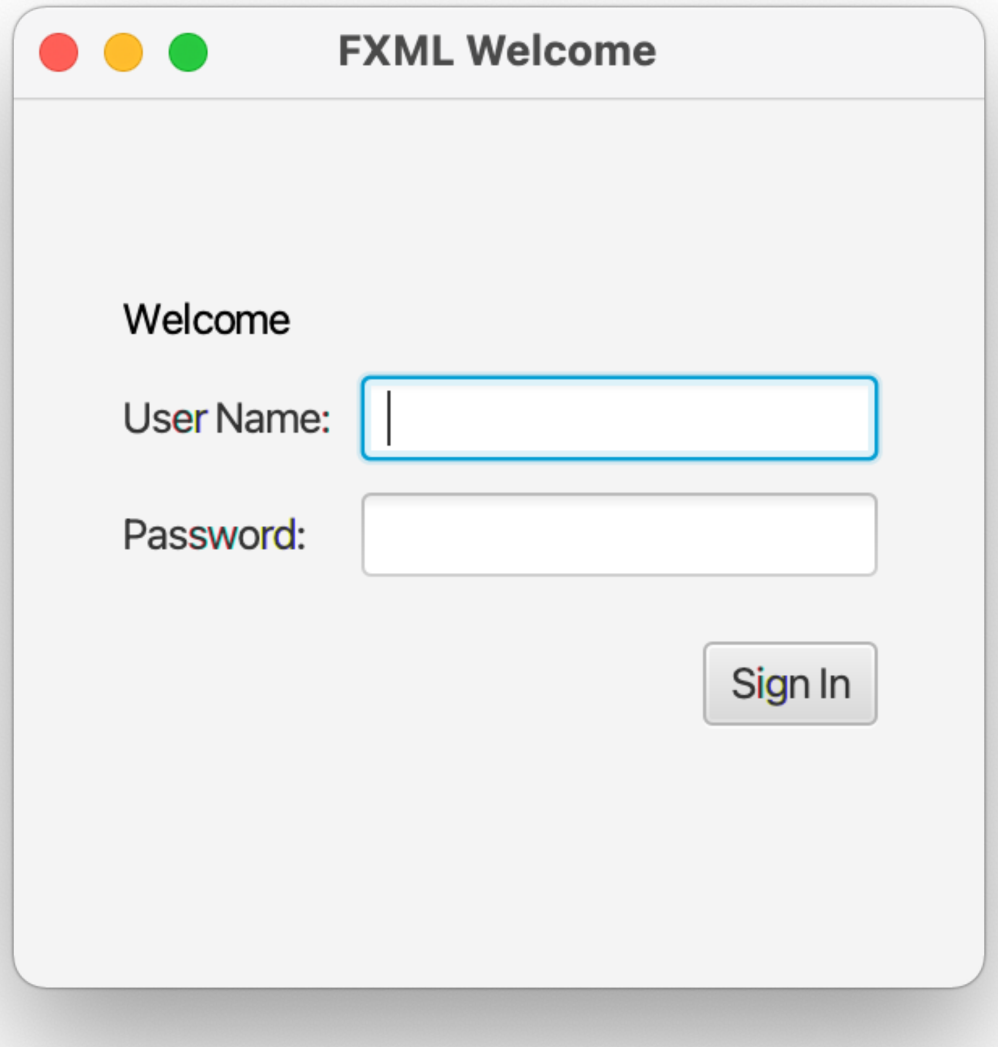
- **Written in Java** and platform-independent
- **FXML** enables developers to create a user interface in a JavaFX application separately from implementing the application logic
- **Scene builder**: drag and drop UI components
- Swing interoperability
- Built-in controls
- CSS support
- Canvas
- Printing API

FXML

- XML-based language that provides the structure for building a user interface separate from the application logic of your code
- https://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html



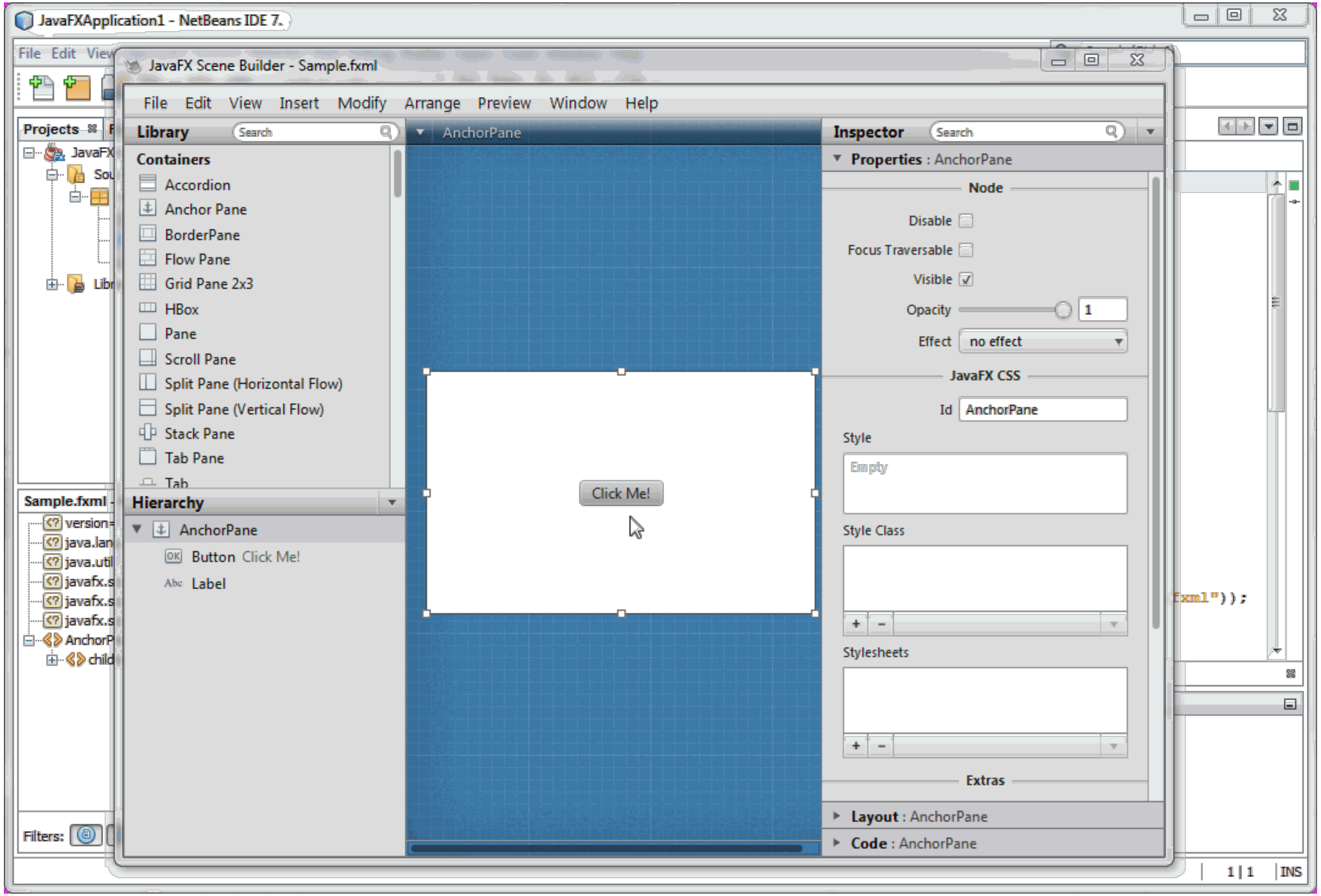
```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
<GridPane fx:controller="de.tum.in.ase.SignInController" xmlns:fx="http://javafx.com/fxml"
                    alignment="center" hgap="10" vgap="10">
    <padding><Insets top="25" right="25" bottom="10" left="25"/></padding>
    <Text text="Welcome" GridPane.columnIndex="0" GridPane.rowIndex="0" GridPane.columnSpan="2"/>
    <Label text="User Name:" GridPane.columnIndex="0" GridPane.rowIndex="1"/>
    <TextField GridPane.columnIndex="1" GridPane.rowIndex="1"/>
    <Label text="Password:" GridPane.columnIndex="0" GridPane.rowIndex="2"/>
    <PasswordField fx:id="passwordField" GridPane.columnIndex="1" GridPane.rowIndex="2"/>
    <HBox spacing="10" alignment="bottom_right" GridPane.columnIndex="1" GridPane.rowIndex="4">
        <Button text="Sign In" onAction="#handleSubmitButtonAction"/>
    </HBox>
    <Text fx:id="actiontarget" GridPane.columnIndex="1" GridPane.rowIndex="6"/>
</GridPane>
```



Scene builder

- Visual layout tool that lets developers quickly design JavaFX user interfaces without coding
- Developers can drag and drop UI components to a work area, modify their properties, and apply style sheets
- The FXML code for the layout that they are creating is automatically generated in the background
- The result is an FXML file that can then be combined with a Java project by binding the UI to the application’s logic
- https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html

Scene builder



Disable ☐

Focus Traversable ☐

Visible ☑

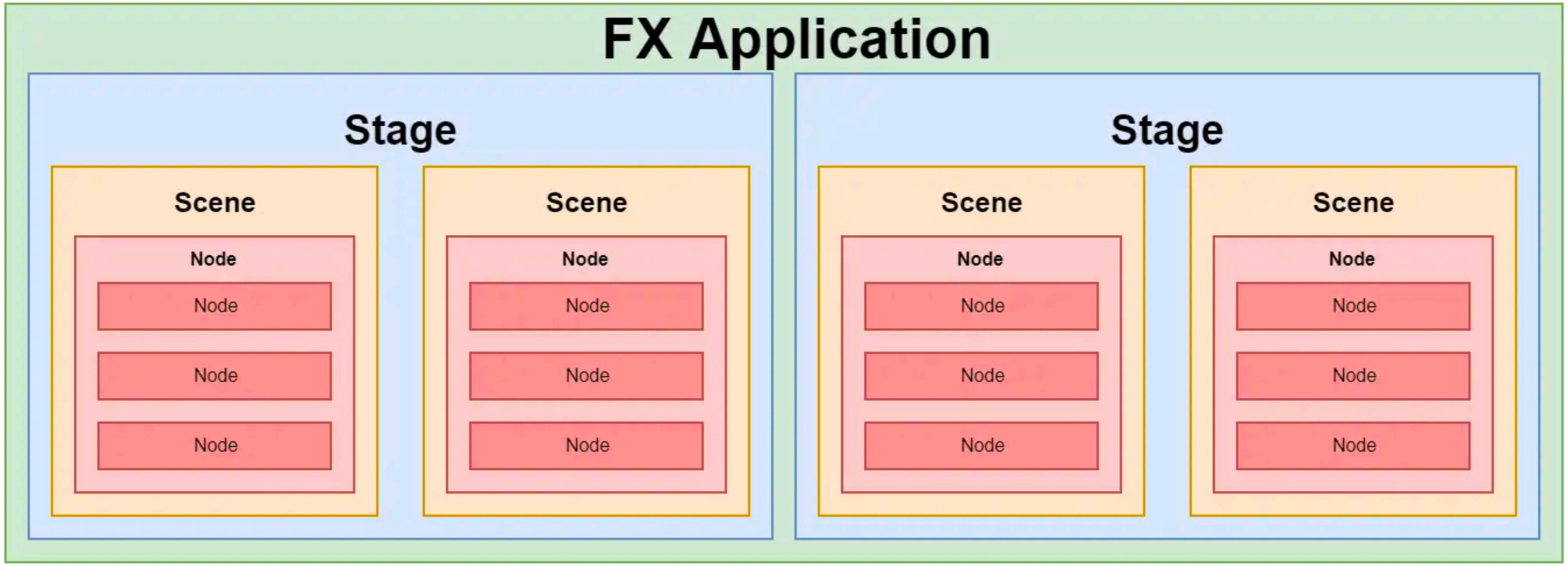
Opacity 1

Effect no effect

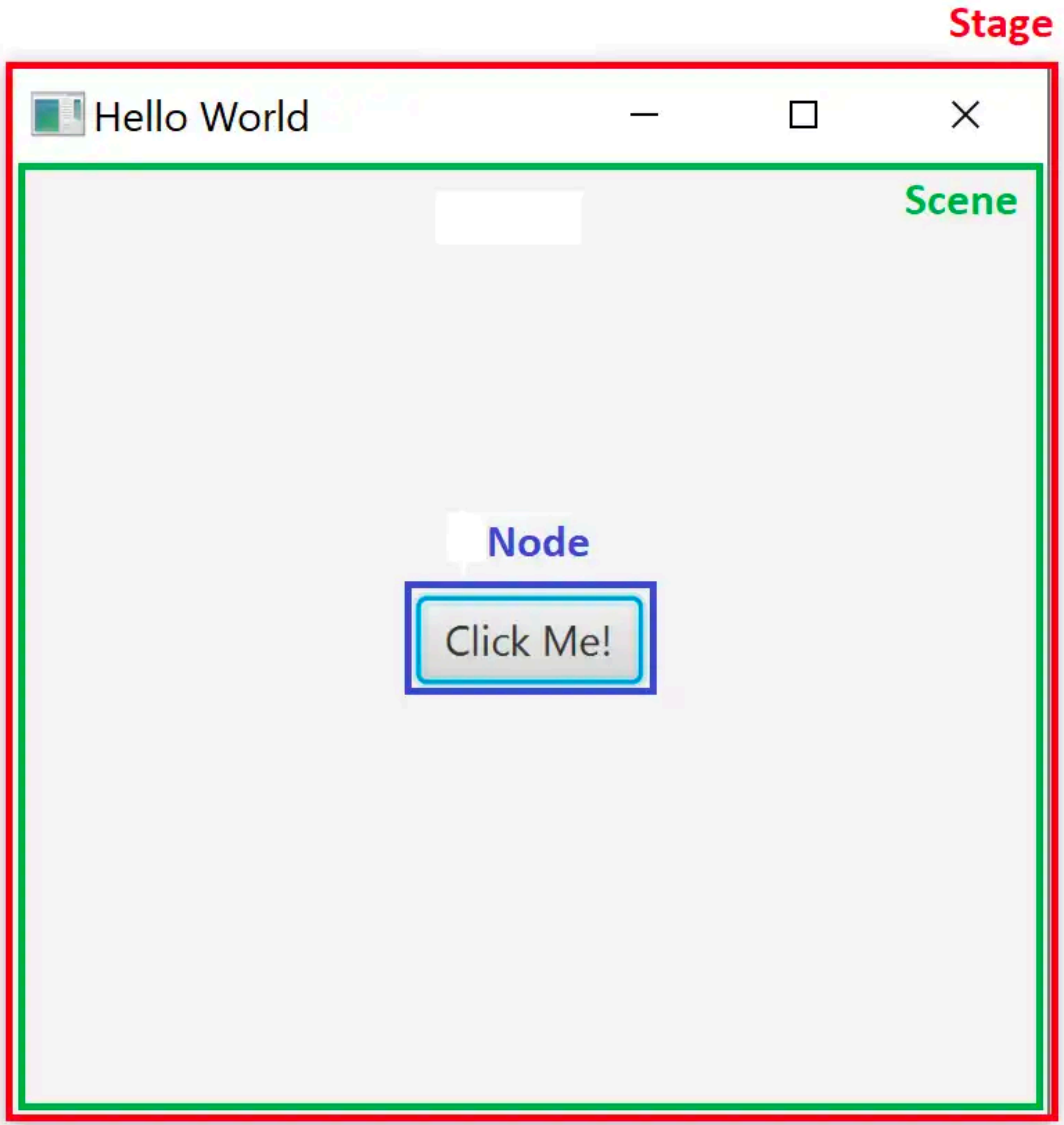
Id AnchorPane

Style: Empty

JavaFX application

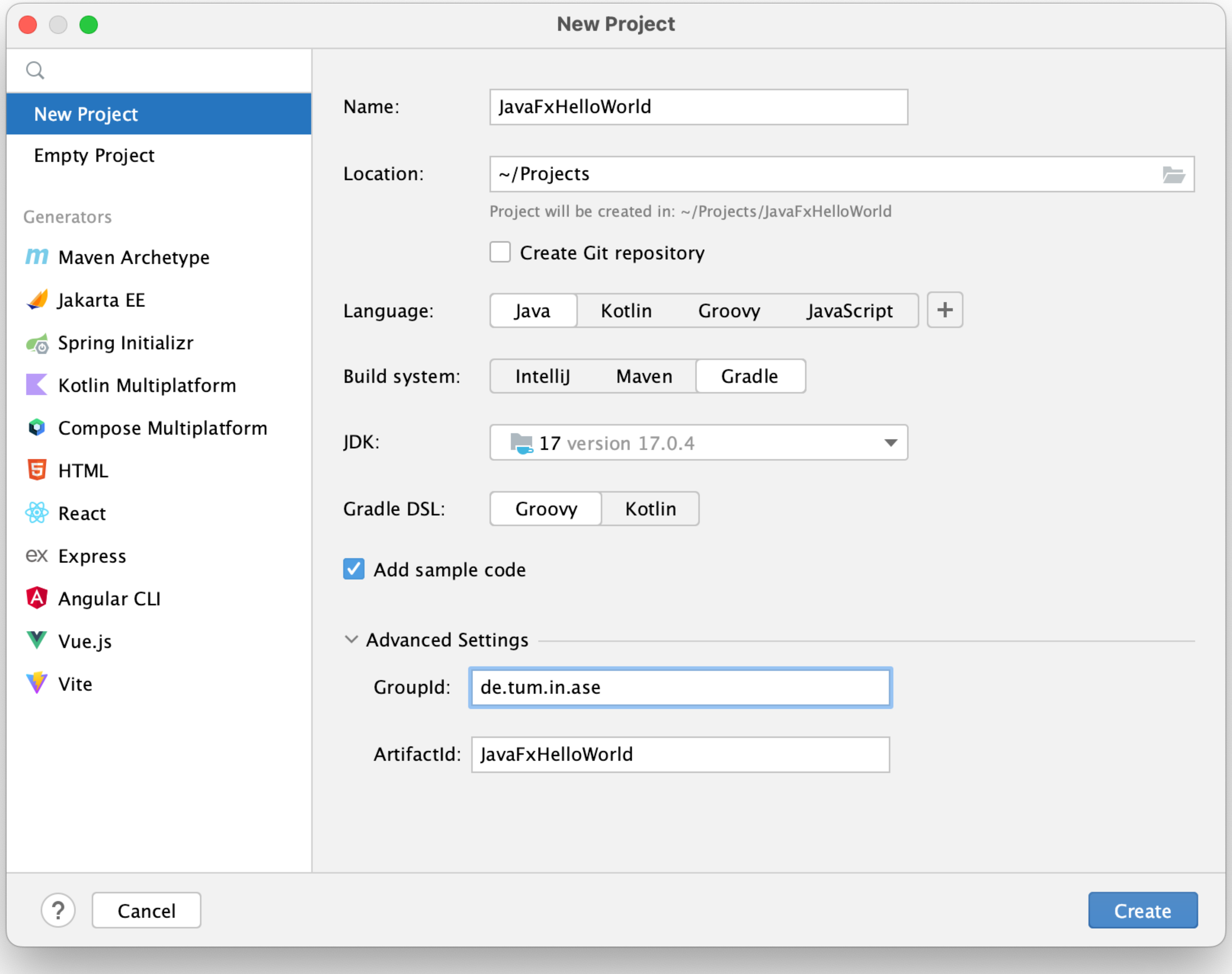


JavaFX application



Interactive tutorial: create a simple JavaFX application

- Create a new Java Gradle project in IntelliJ



New Project

Name: JavaFxHelloWorld

Location: ~/Projects

Project will be created in: ~/Projects/JavaFxHelloWorld

☐ Create Git repository

Language: Java | Kotlin | Groovy | JavaScript | +

Build system: IntelliJ | Maven | Gradle

JDK: 17 version 17.0.4

Gradle DSL: Groovy | Kotlin

☑ Add sample code

Advanced Settings

GroupId: de.tum.in.ase

ArtifactId: JavaFxHelloWorld

Interactive tutorial: create a simple JavaFX application

- Open the **`build.gradle`** and insert the following code

```
plugins {
    id 'application'
    id 'org.openjfx.javafxplugin' version '0.1.0'
    id 'java'
}
java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}
version = '1.0.0'
compileJava.options.encoding = 'UTF-8'

repositories {
    mavenCentral()
}

javafx {
    version = '17.0.17'
    modules = [ 'javafx.base', 'javafx.controls', 'javafx.fxml', 'javafx.media' ]
}

application {
    mainModule = 'JavaFxHelloWorld.main'
    mainClass = 'de.tum.in.ase.JavaFxHelloWorld'
}
```

Interactive tutorial: create a simple JavaFX application

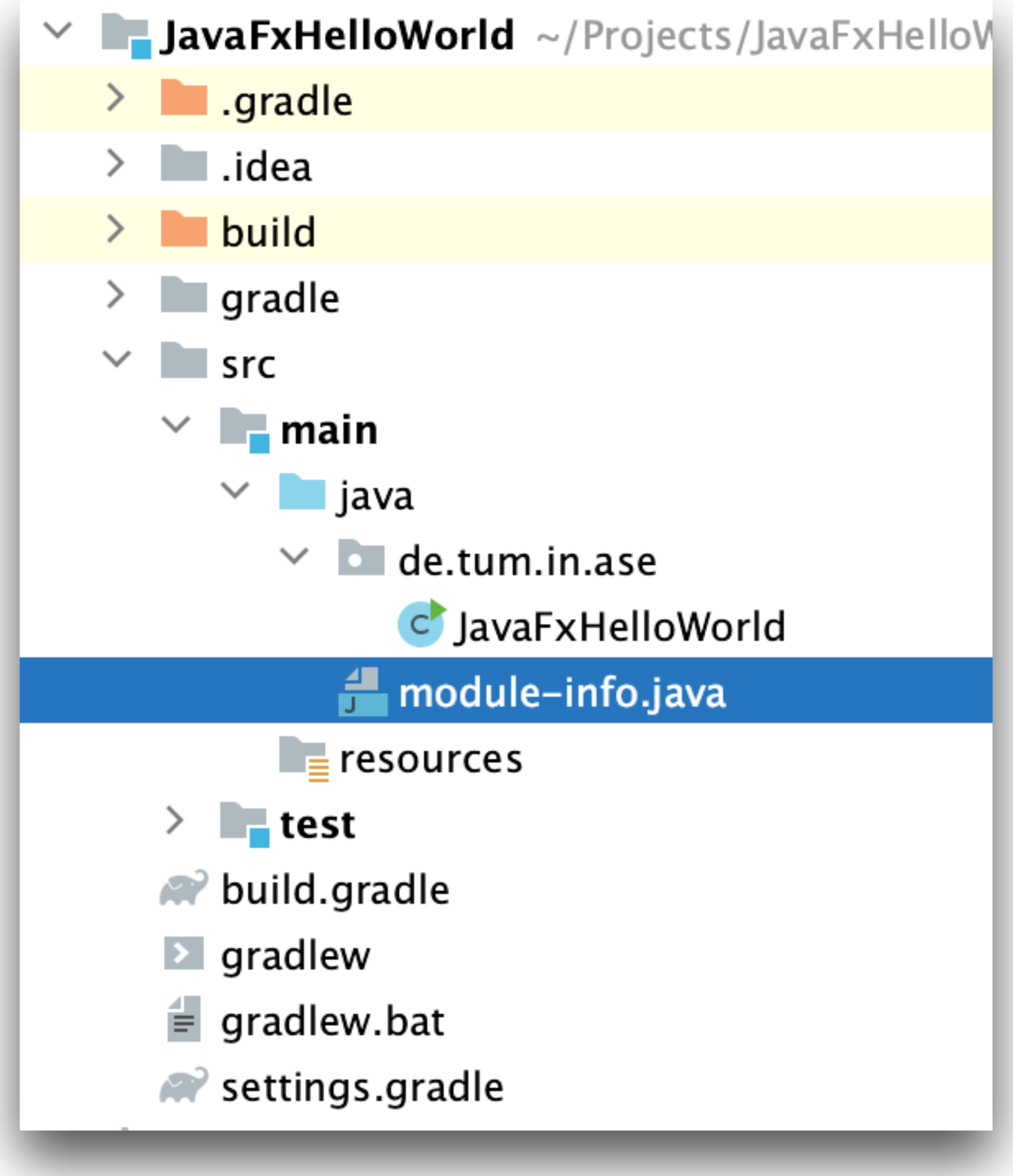
- Create a new package **`de.tum.in.ase`** and add a new class **`JavaFxHelloWorld`**

```java
package de.tum.in.ase;

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

public class JavaFxHelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        primaryStage.show();
    }

    public static void main(String[] args) {
        Application.launch(JavaFxHelloWorld.class, args);
    }
}
```

Interactive tutorial: create a simple JavaFX application

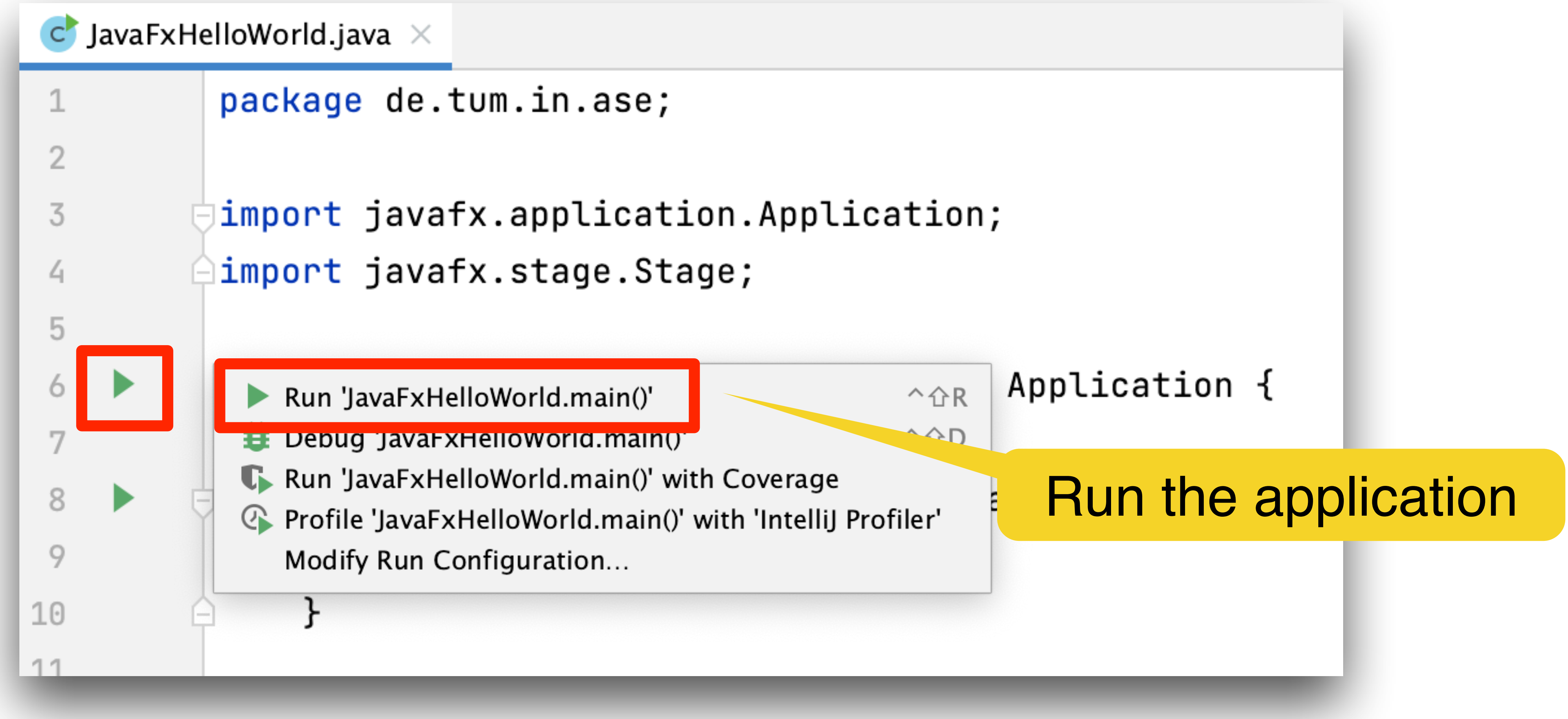
- Create **`module-info.java`** in the root package

```
module JavaFxHelloWorld.main {
    requires javafx.graphics;
    requires javafx.fxml;
    requires javafx.controls;
    opens de.tum.in.ase to javafx.graphics, javafx.fxml;
    exports de.tum.in.ase;
}
```



```
JavaFxHelloWorld ~/Projects/JavaFxHelloW
  .gradle
  .idea
  build
  gradle
  src
    main
      java
        de.tum.in.ase
          JavaFxHelloWorld
          module-info.java
      resources
    test
  build.gradle
  gradlew
  gradlew.bat
  settings.gradle
```

- Run the application



JavaFxHelloWorld.java

```
package de.tum.in.ase;

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

                                                   Application {



    }
```

Run 'JavaFxHelloWorld.main()' ^⇧R
Debug 'JavaFxHelloWorld.main()' ^⇧D
Run 'JavaFxHelloWorld.main()' with Coverage
Profile 'JavaFxHelloWorld.main()' with 'IntelliJ Profiler'
Modify Run Configuration...

Run the application

The lifecycle of a JavaFX application

- Entry point of JavaFX applications: the class **`Application`**
- The JavaFX runtime does the following, in order, when an application is launched
  1. It creates an instance of the specified **`Application`** class
  2. It calls the **`init()`** method of the **`Application`** class
  3. It calls the **`start()`** method
  4. The application is visible in the foreground, the runtime waits for the application to finish
- The application exits when one of the following occurs
  - The app calls **`Platform.exit()`**
  - The last window of the app is closed
  - Before exiting, the **`stop()`** method of **`Application`** class is called
- You can override **`init()`**, **`start()`** and **`stop()`** to perform any initialization and cleanup of resources used by your application

Interactive tutorial: FXML

- Change **`JavaFxHelloWorld`**

```java
package de.tum.in.ase;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class JavaFxHelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getClassLoader().getResource("example.fxml"));
        Scene scene = new Scene(root, 300, 275);
        primaryStage.setTitle("FXML Welcome");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Interactive tutorial: FXML

- Add a new class **`SignInController`** in the package **`de.tum.in.ase`**

```java
package de.tum.in.ase;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.PasswordField;
import javafx.scene.text.Text;

public class SignInController {

    @FXML public PasswordField passwordField;

    @FXML private Text actiontarget;

    @FXML protected void handleSubmitButtonAction(ActionEvent event) {
        actiontarget.setText("Sign in button pressed");
    }
}
```

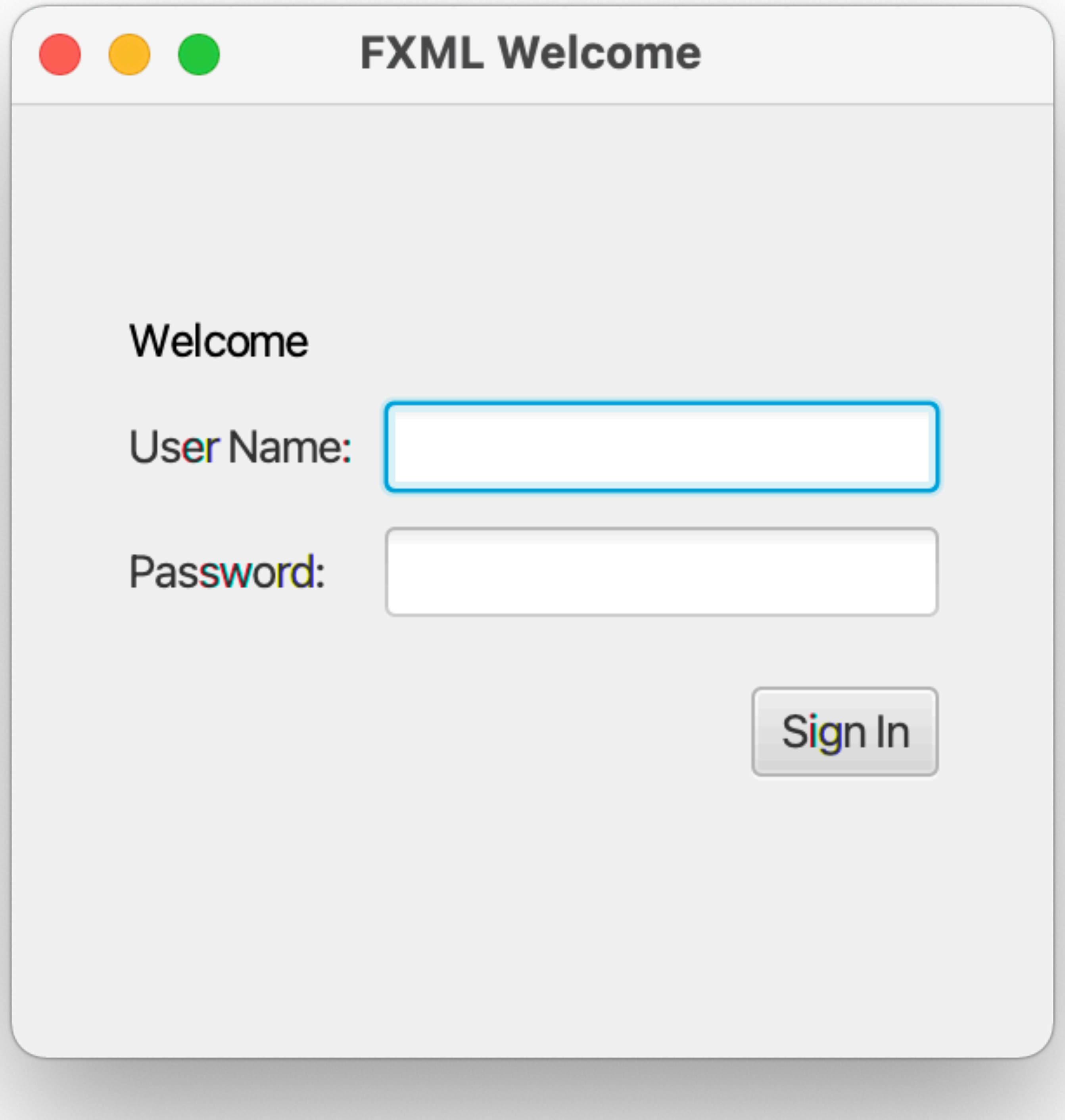
Interactive tutorial: FXML

- Create a new file **`example.fxml`** in the resources folder

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
<GridPane fx:controller="de.tum.in.ase.SignInController" xmlns:fx="http://javafx.com/fxml"
                         alignment="center" hgap="10" vgap="10">
    <padding><Insets top="25" right="25" bottom="10" left="25"/></padding>
    <Text text="Welcome" GridPane.columnIndex="0" GridPane.rowIndex="0" GridPane.columnSpan="2"/>
    <Label text="User Name:" GridPane.columnIndex="0" GridPane.rowIndex="1"/>
    <TextField GridPane.columnIndex="1" GridPane.rowIndex="1"/>
    <Label text="Password:" GridPane.columnIndex="0" GridPane.rowIndex="2"/>
    <PasswordField fx:id="passwordField" GridPane.columnIndex="1" GridPane.rowIndex="2"/>
    <HBox spacing="10" alignment="bottom_right" GridPane.columnIndex="1" GridPane.rowIndex="4">
        <Button text="Sign In" onAction="#handleSubmitButtonAction"/>
    </HBox>
    <Text fx:id="actiontarget" GridPane.columnIndex="1" GridPane.rowIndex="6"/>
</GridPane>
```

Interactive tutorial: FXML

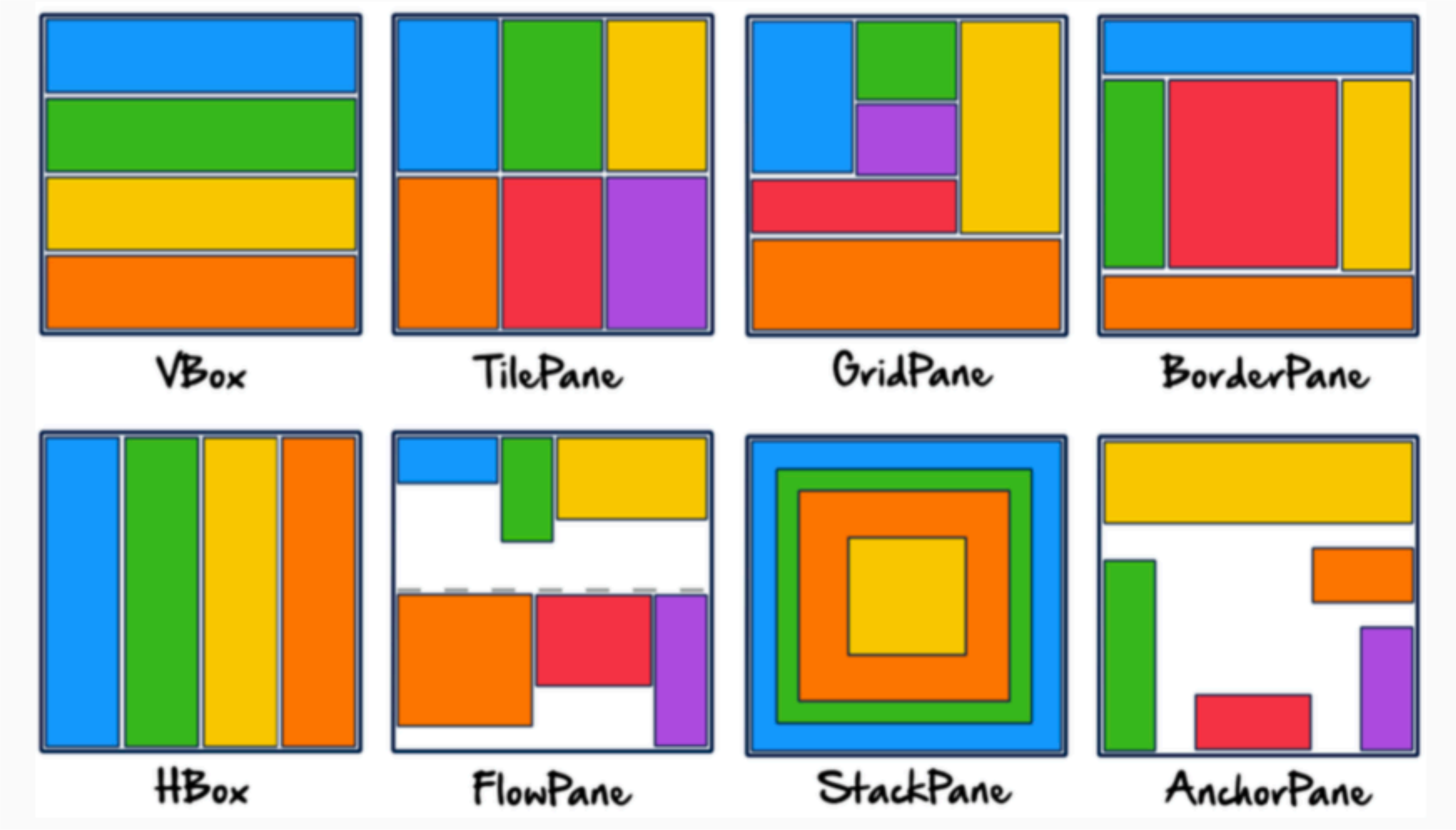
- Run the application again



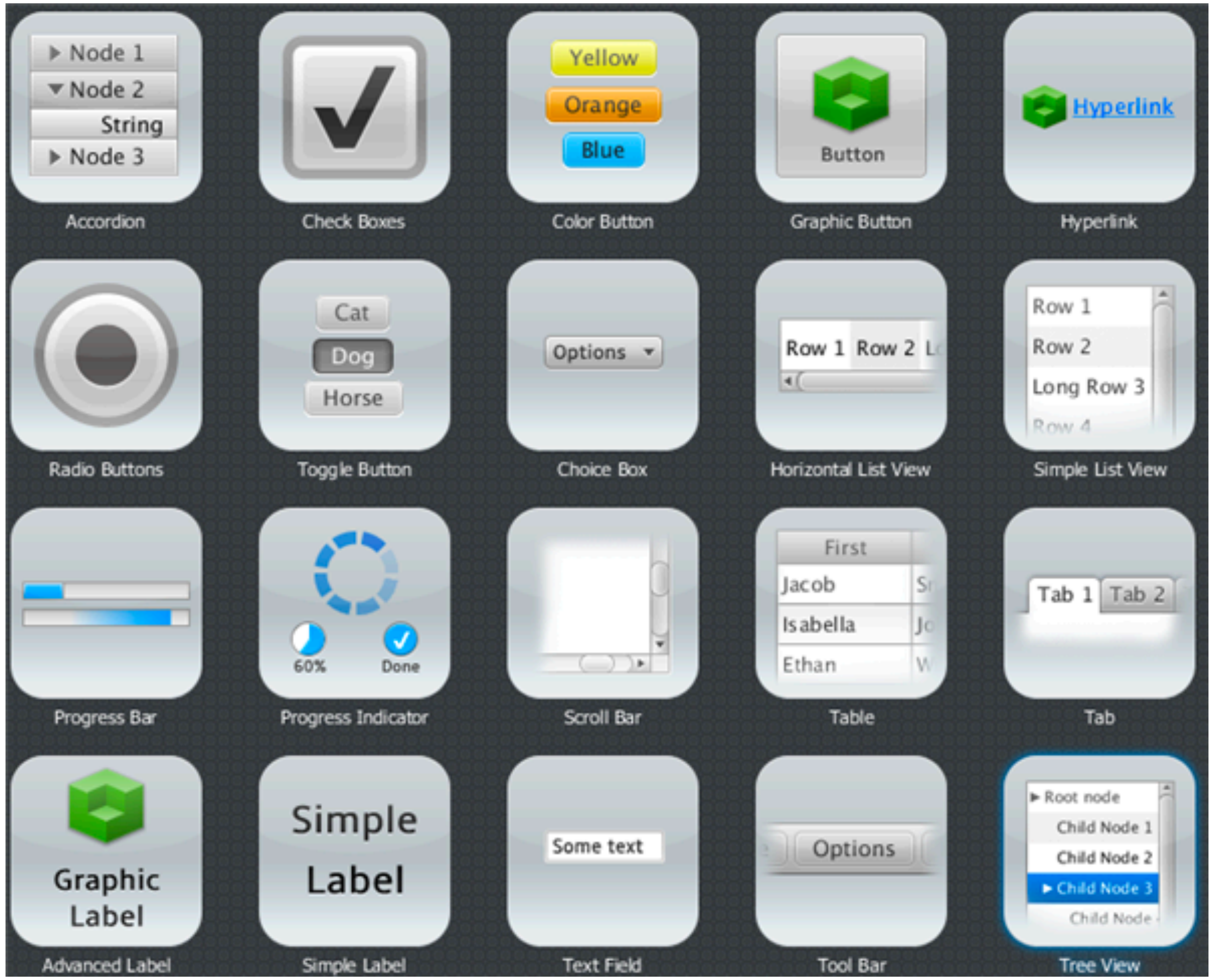
Overview of JavaFX user interface concepts

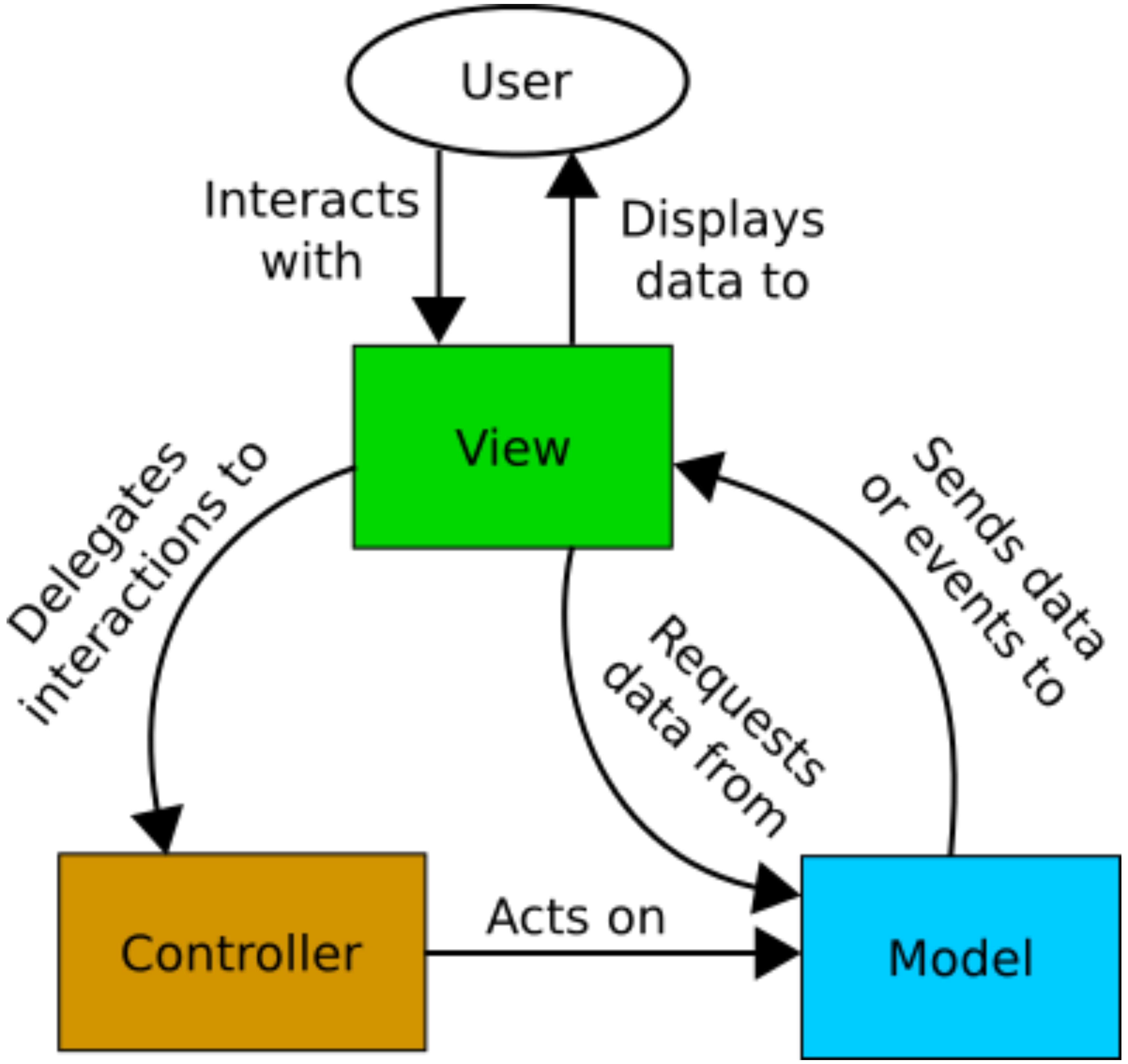
- Layouts
- Controls for user input
- Shapes
- Styling

Layouts

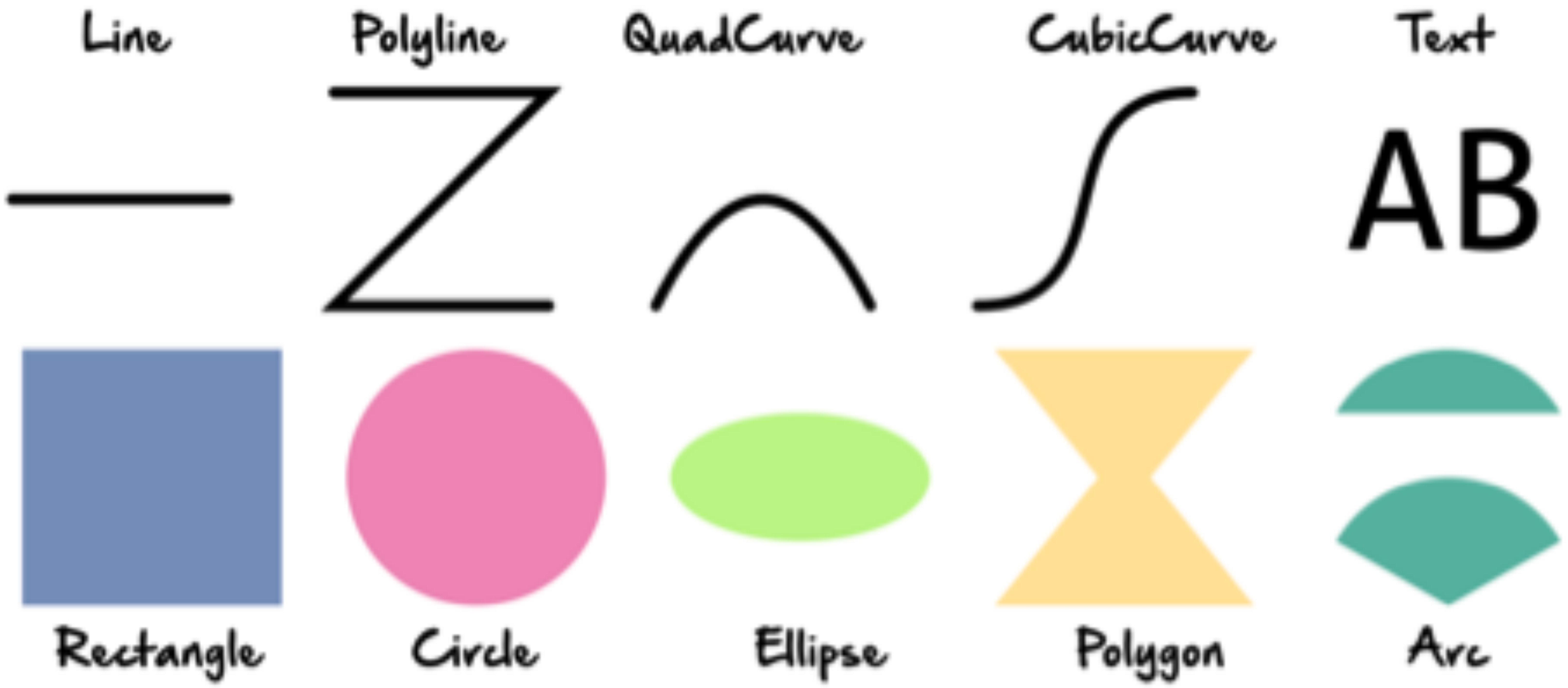

VBox
TilePane
GridPane
BorderPane
HBox
FlowPane
StackPane
AnchorPane

Controls for user input

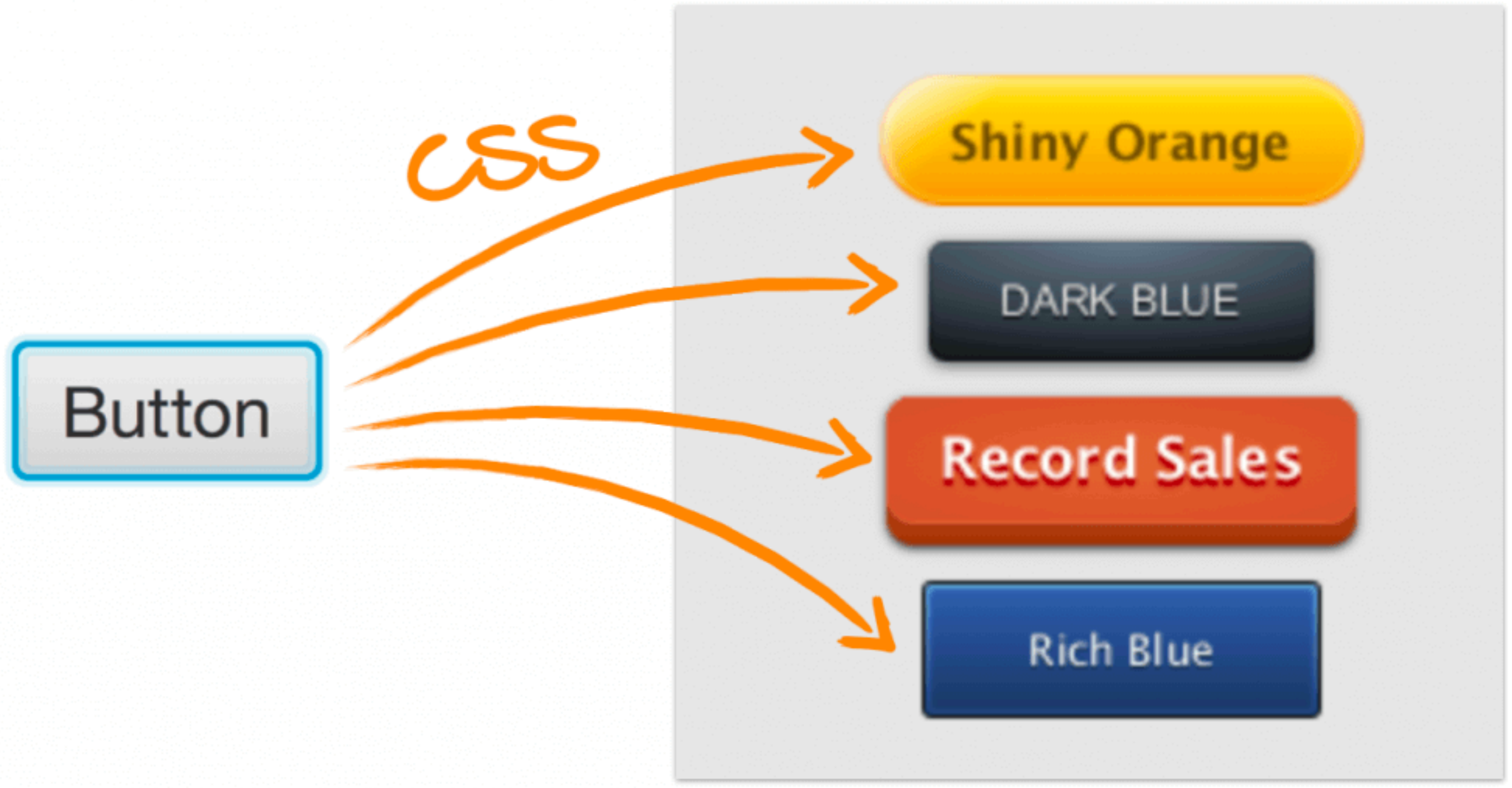




Shapes


Line
Polyline
QuadCurve
CubicCurve
Text
AB
Rectangle
Circle
Ellipse
Polygon
Arc

Styling



Outline

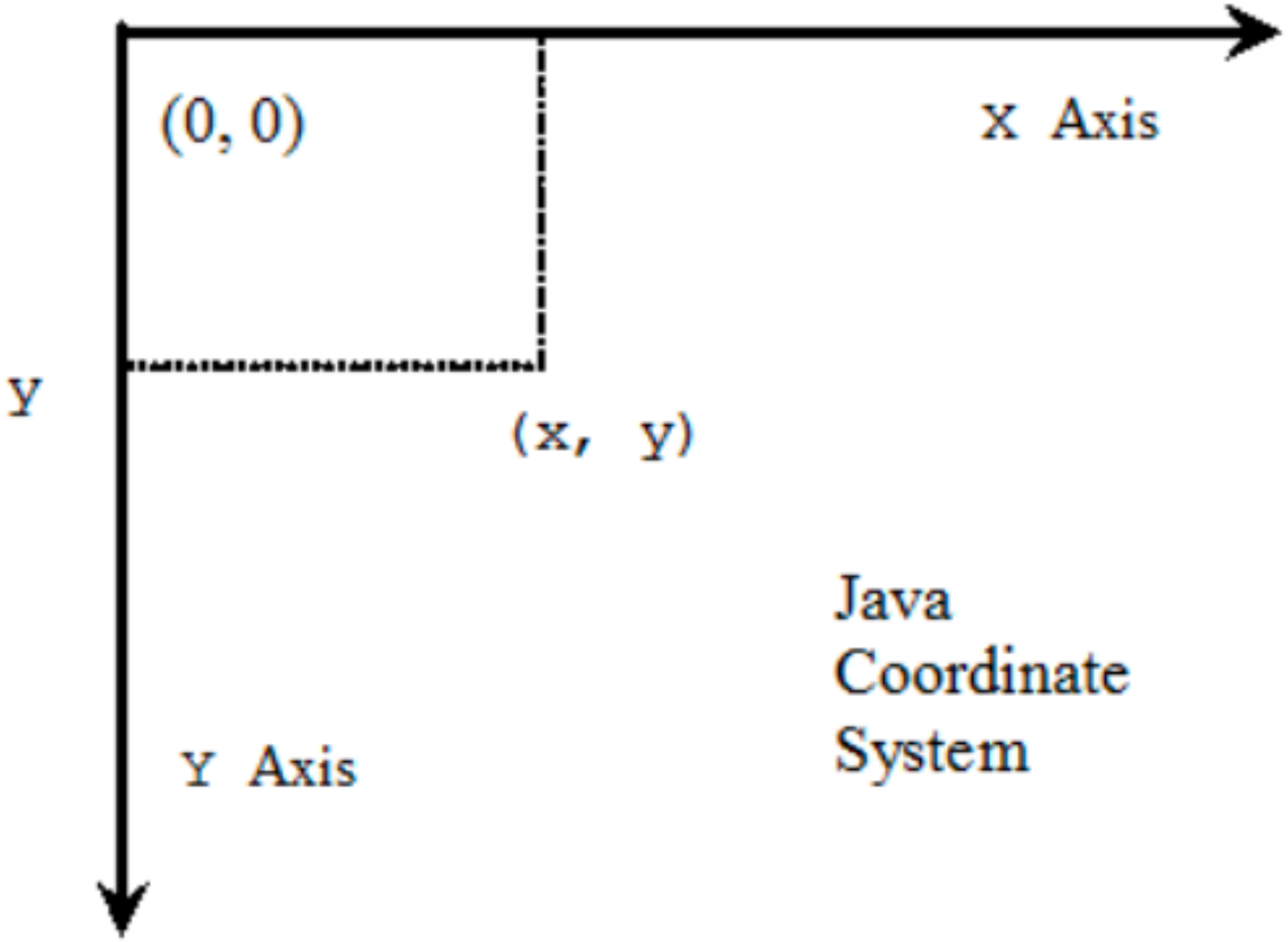
- Usability
- JavaFX

➡ Layout

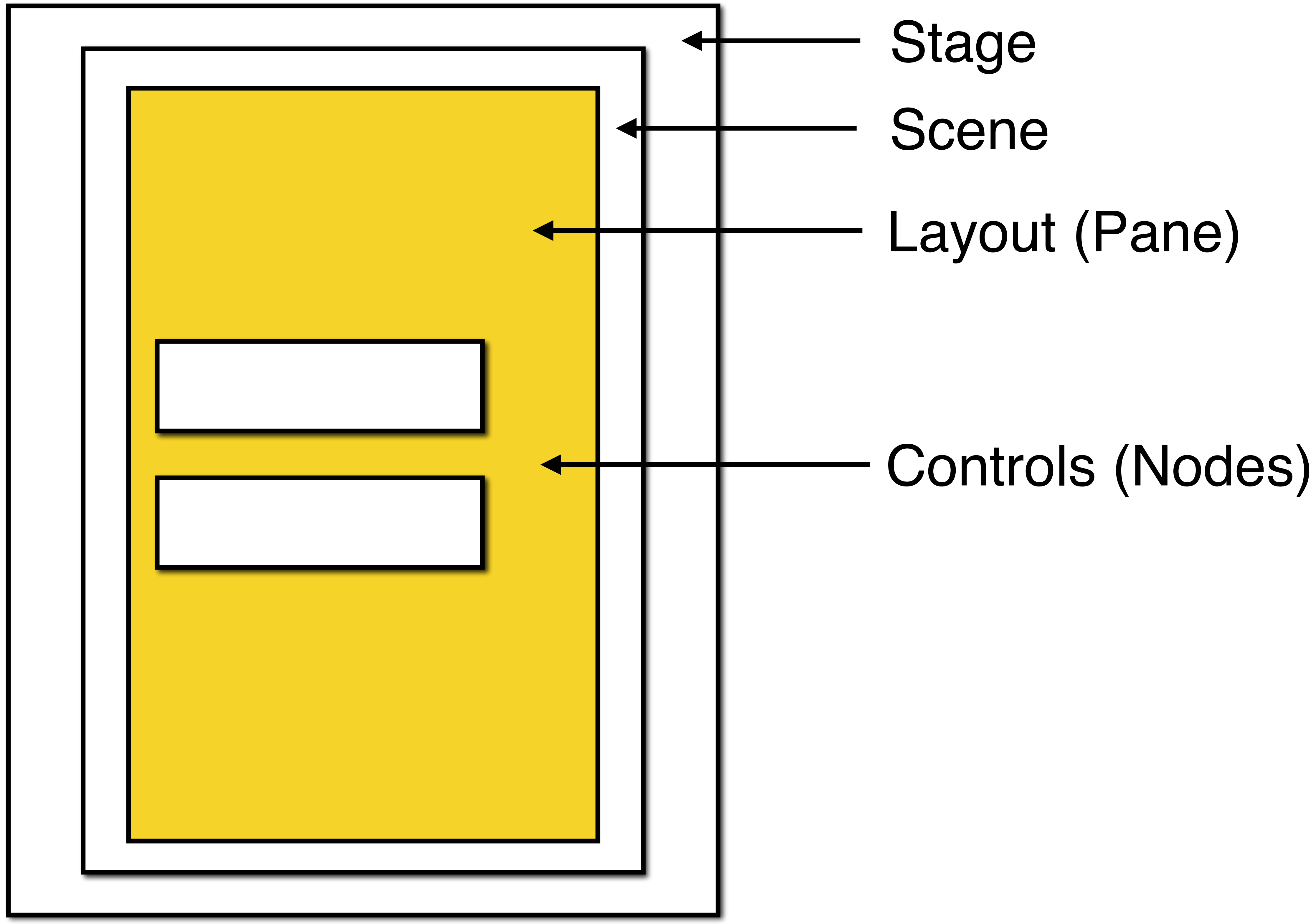
- User input
- Shapes
- Styling

Coordinate system

- Starts from the left upper corner in JavaFX



Layout

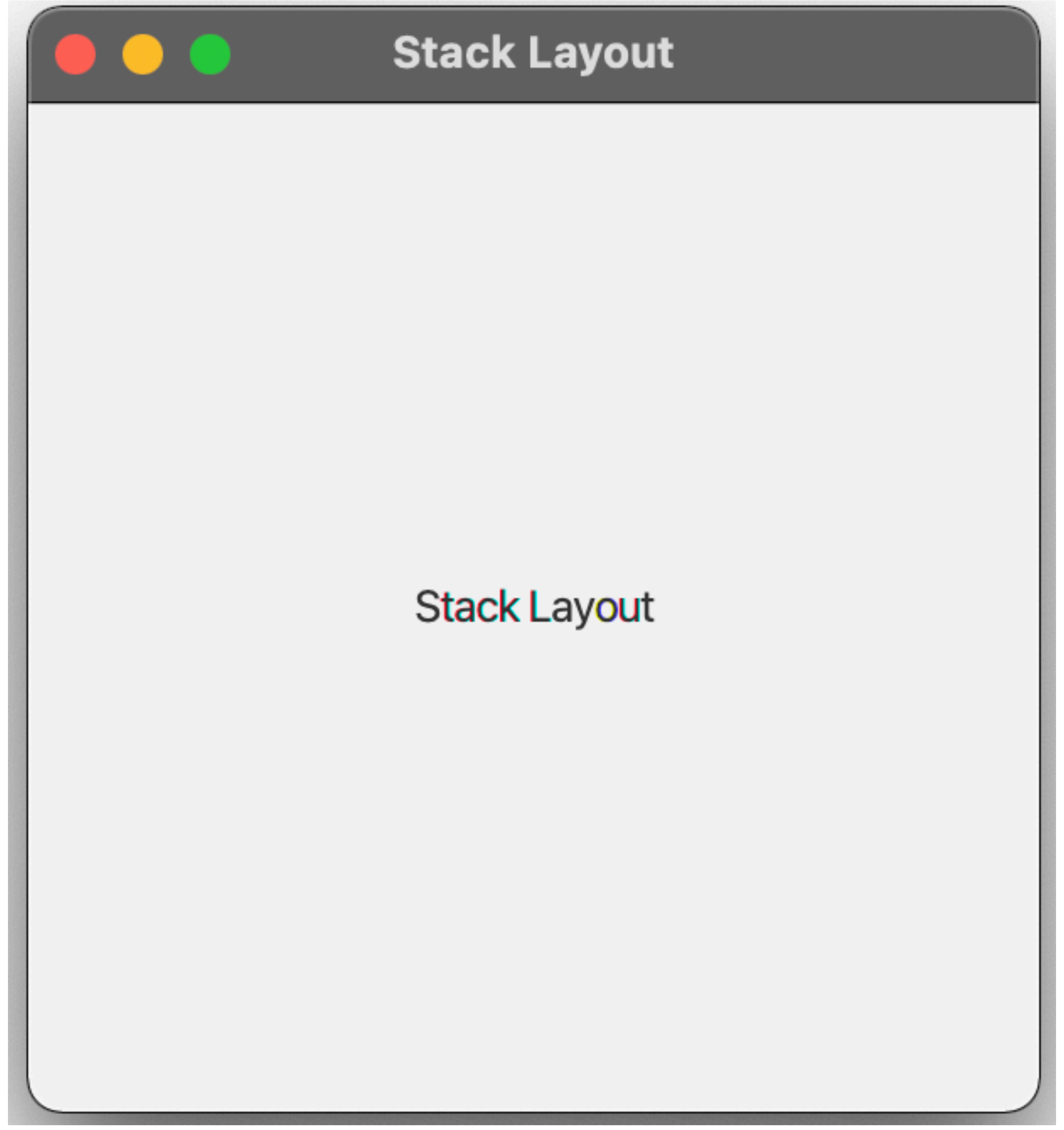


- The class **`Layout`** decides how sub nodes (e.g. buttons) are distributed inside the window
- Decides in which position the buttons and other components are positioned
  - Example: if controls are aligned, e.g. in the form of a matrix
  - Example: which controls become smaller/ larger when the window is resized, etc.
- JavaFX provides many types of layouts for organizing nodes in a scene

Stack layout

- Places the controls on top of each other in the center of the layout

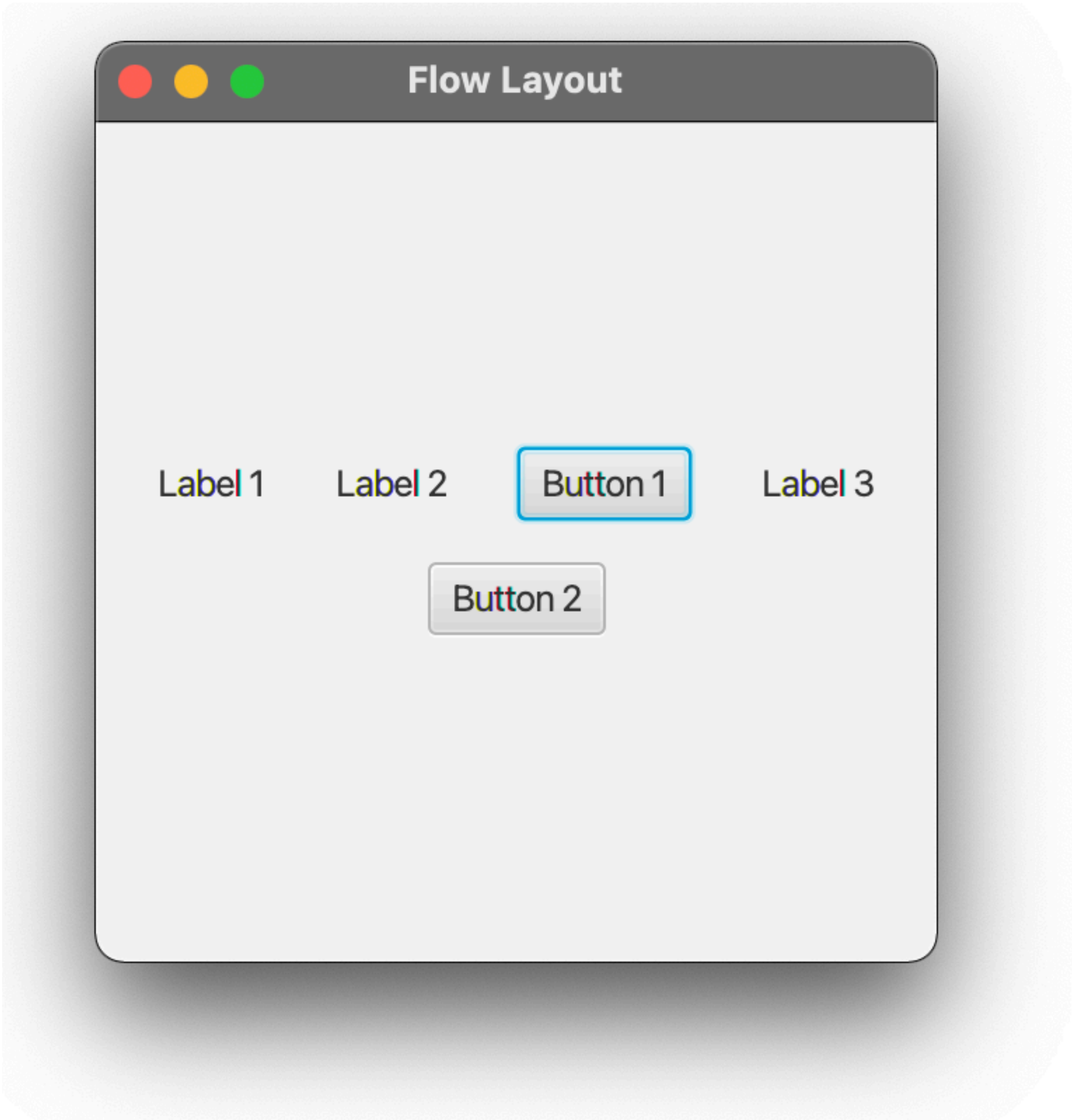
```
public class StackLayoutApp extends Application {
    @Override
    public void start(Stage stage) {
        StackPane spane = new StackPane();
        spane.getChildren().add(new Label("Stack Layout"));

        Scene scene = new Scene(spane, 300, 300);
        stage.setTitle("Stack Layout");
        stage.setScene(scene);
        stage.show();
    }
}
```



Flow layout

- Places the controls row-by-row horizontally or column-by-column vertically

```
public class FlowLayoutApp extends Application {
    @Override
    public void start(Stage stage) {
        FlowPane fpane = new FlowPane();
        fpane.setHgap(25);
        fpane.setVgap(15);
        fpane.setAlignment(Pos.CENTER);
        fpane.getChildren().addAll(new Label("Label 1"),
                                   new Label("Label 2"),
                                   new Button("Button 1"),
                                   new Label("Label 3"),
                                   new Button("Button 2"));
        Scene scene = new Scene(fpane, 300, 300);
        stage.setTitle("Flow Layout");
        stage.setScene(scene);
        stage.show();
    }
}
```



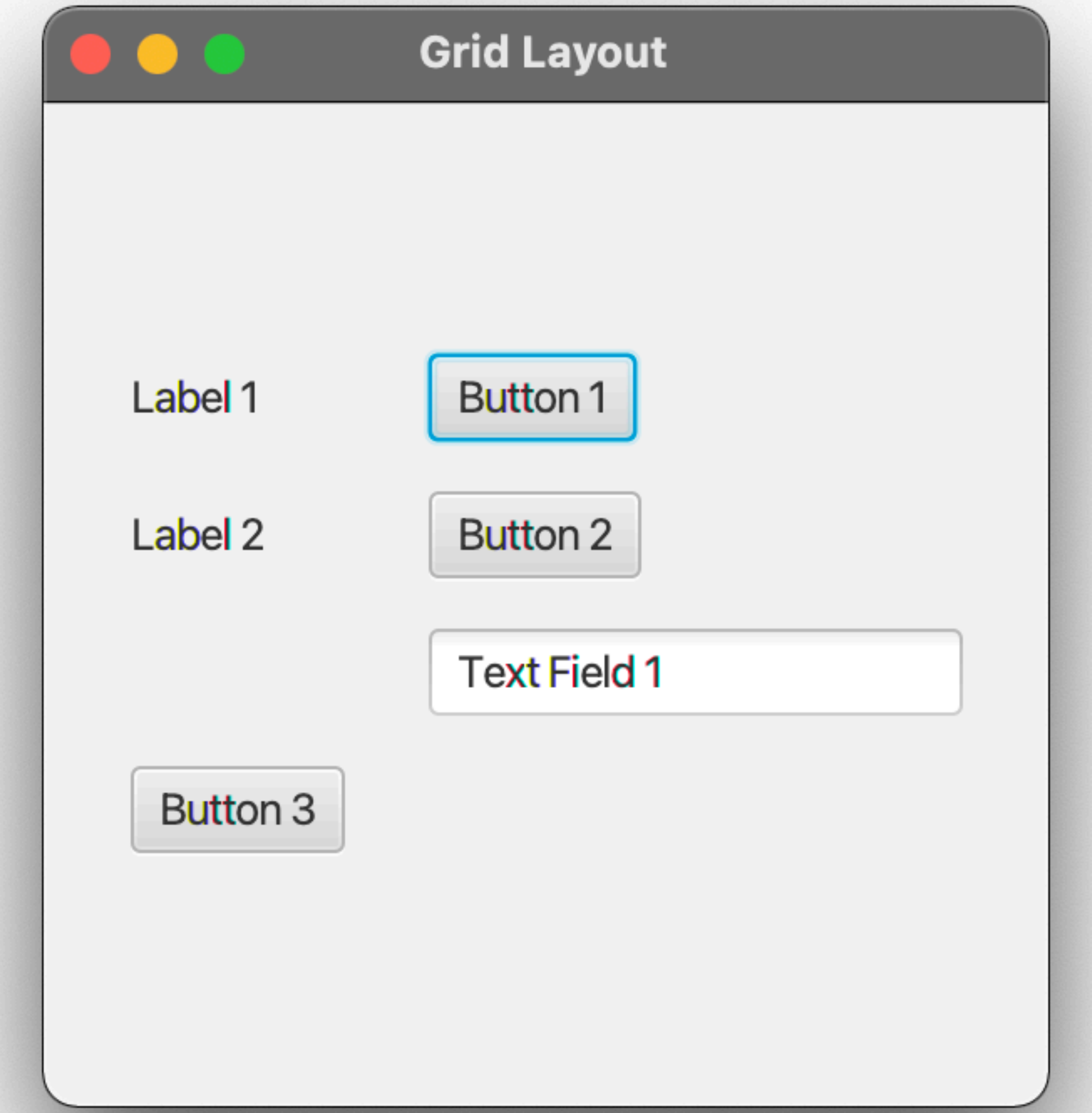
Grid layout

- Places the controls in different cells in a two-dimensional grid

```
public class GridLayoutApp extends Application {
    @Override
    public void start(Stage stage) {
        GridPane gpane = new GridPane();
        gpane.setHgap(25);
        gpane.setVgap(15);
        gpane.setAlignment(Pos.CENTER);
        gpane.add(new Label("Label 1"), 0, 0);
        gpane.add(new Button("Button 1"), 1, 0);
        gpane.add(new Label("Label 2"), 0, 1);
        gpane.add(new Button("Button 2"), 1, 1);
        gpane.add(new TextField("Text Field 1"), 1, 2);
        gpane.add(new Button("Button 3"), 0, 3);
        Scene scene = new Scene(gpane, 300, 300);
        stage.setTitle("Grid Layout");
        stage.setScene(scene);
        stage.show();
    }
}
```

You can locate the controls in different cells

Note that (0,0) is in this example the top left corner in the grid

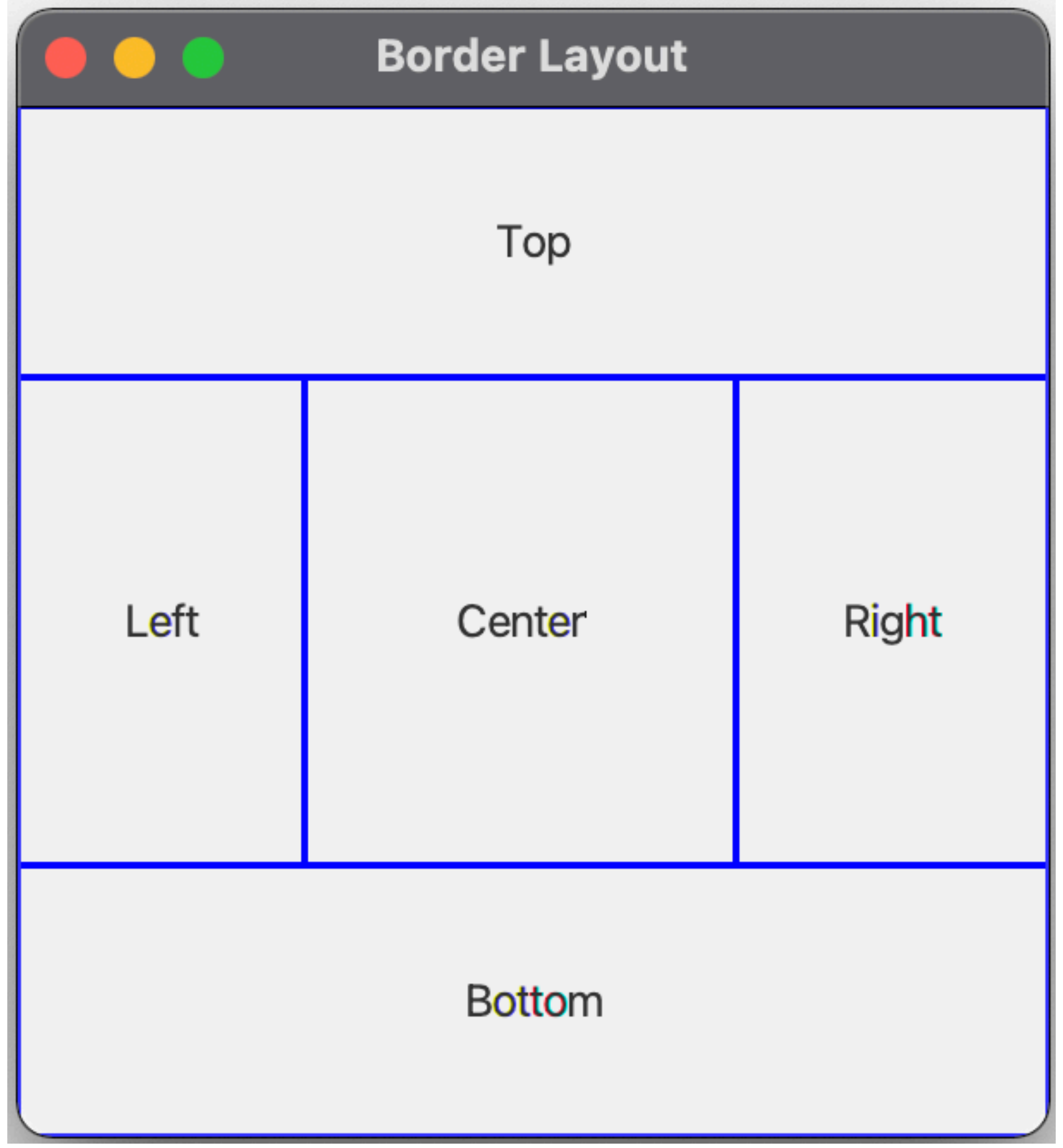


Border layout

- Places the controls in the top, right, center, left, and bottom regions

```
public class BorderLayoutApp extends Application {
    @Override
    public void start(Stage stage) {
        BorderPane bpane = new BorderPane();
        bpane.setTop(new CustomLayout("Top"));
        bpane.setBottom(new CustomLayout("Bottom"));
        bpane.setRight(new CustomLayout("Right"));
        bpane.setLeft(new CustomLayout("Left"));
        bpane.setCenter(new CustomLayout("Center"));
        Scene scene = new Scene(bpane, 300, 300);
        stage.setTitle("Border Layout");
        stage.setScene(scene);
        stage.show();
    }

    static class CustomLayout extends StackPane {
        public CustomLayout(String title) {
            getChildren().add(new Label(title));
            setStyle("-fx-border-color: blue");
            setPadding(new Insets(30,30,30,30));
        }
    }
}
```

Extends from **StackPane**



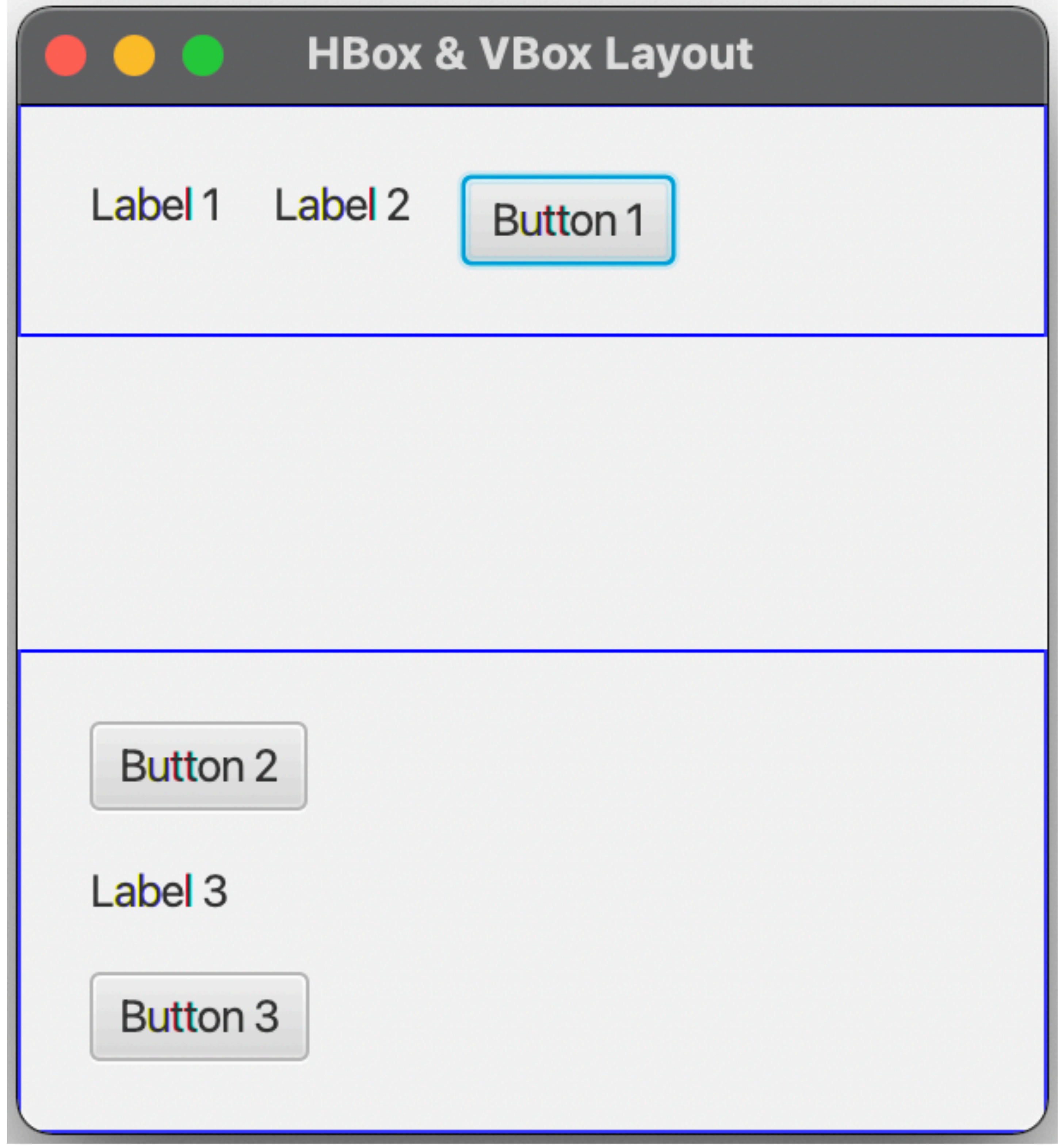
HBox and VBox layout

- Places the controls in a single row (**HBox**) or a single column (**VBox**)

```java
public class BoxLayoutApp extends Application {
    @Override
    public void start(Stage stage) {
        BorderPane layout = new BorderPane();

        HBox hBox = new HBox(15);
        hBox.setPadding(new Insets(20, 20, 20, 20));
        hBox.setStyle("-fx-border-color: blue");
        hBox.getChildren().add(new Label("Label 1"));
        hBox.getChildren().add(new Label("Label 2"));
        hBox.getChildren().add(new Button("Button 1"));
        layout.setTop(hBox);

        VBox vBox = new VBox(15);
        vBox.setStyle("-fx-border-color: blue");
        vBox.setPadding(new Insets(20, 20, 20, 20));
        vBox.getChildren().add(new Button("Button 2"));
        vBox.getChildren().add(new Label("Label 3"));
        vBox.getChildren().add(new Button("Button 3"));
        layout.setBottom(vBox);

        Scene scene = new Scene(layout,300, 300);
        stage.setTitle("HBox & VBox Layout");
        stage.setScene(scene);
        stage.show();
    }
}
```

Places controls in a single row

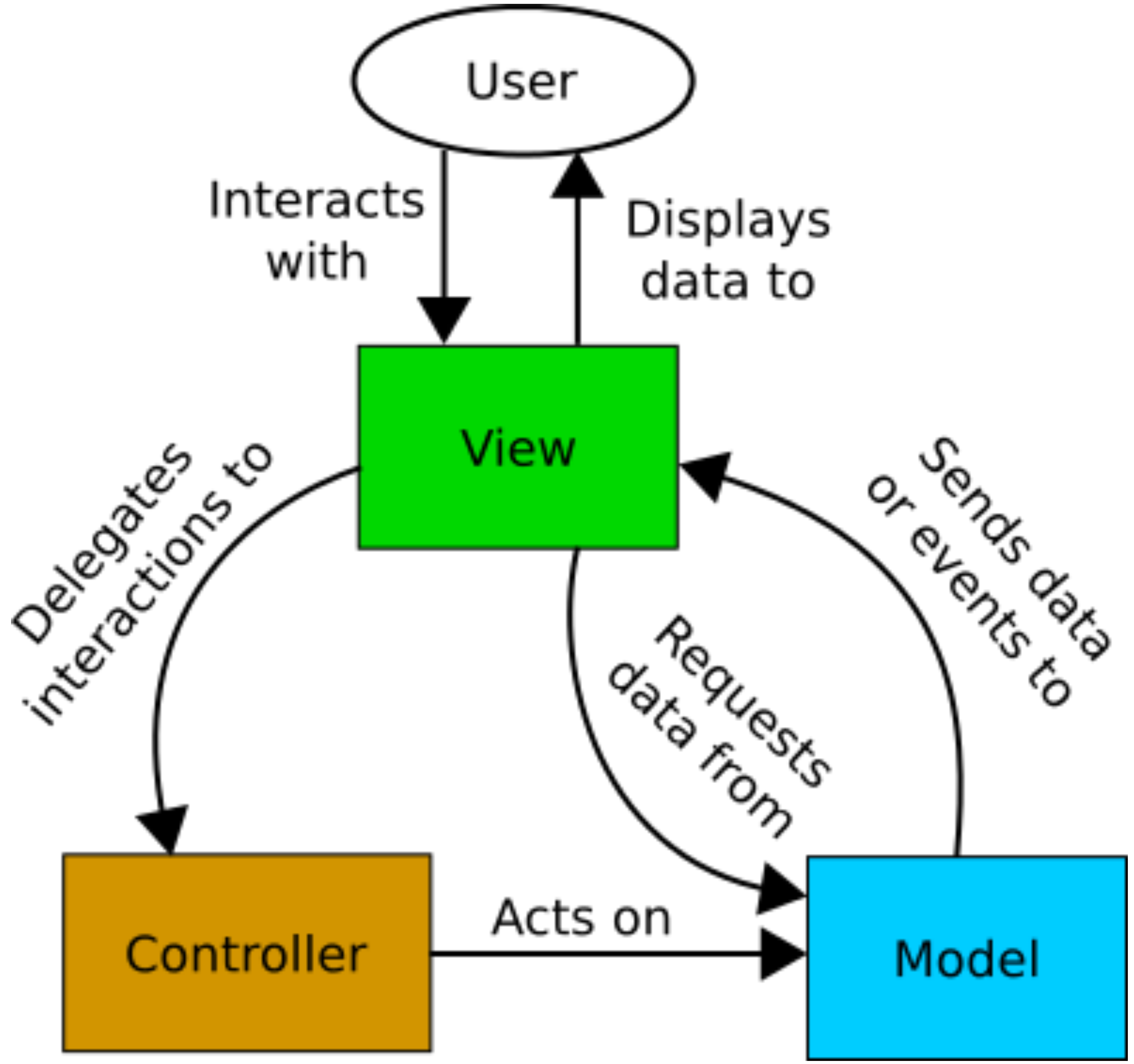
Places controls in a single column



Outline

- Usability
- JavaFX
- Layout
- ➡ User input
- Shapes
- Styling

Model view controller (MVC)



User input

- Controls allow users to provide information to the program: **`TextField`**, **`PasswordField`**, etc.
- There are elements for providing some status to the program such as **`CheckBox`**, **`ChoiceBox`**, etc.
- Some controls allow users to see information: **`Label`**, **`ListView`**, **`TableView`**, etc.

User input

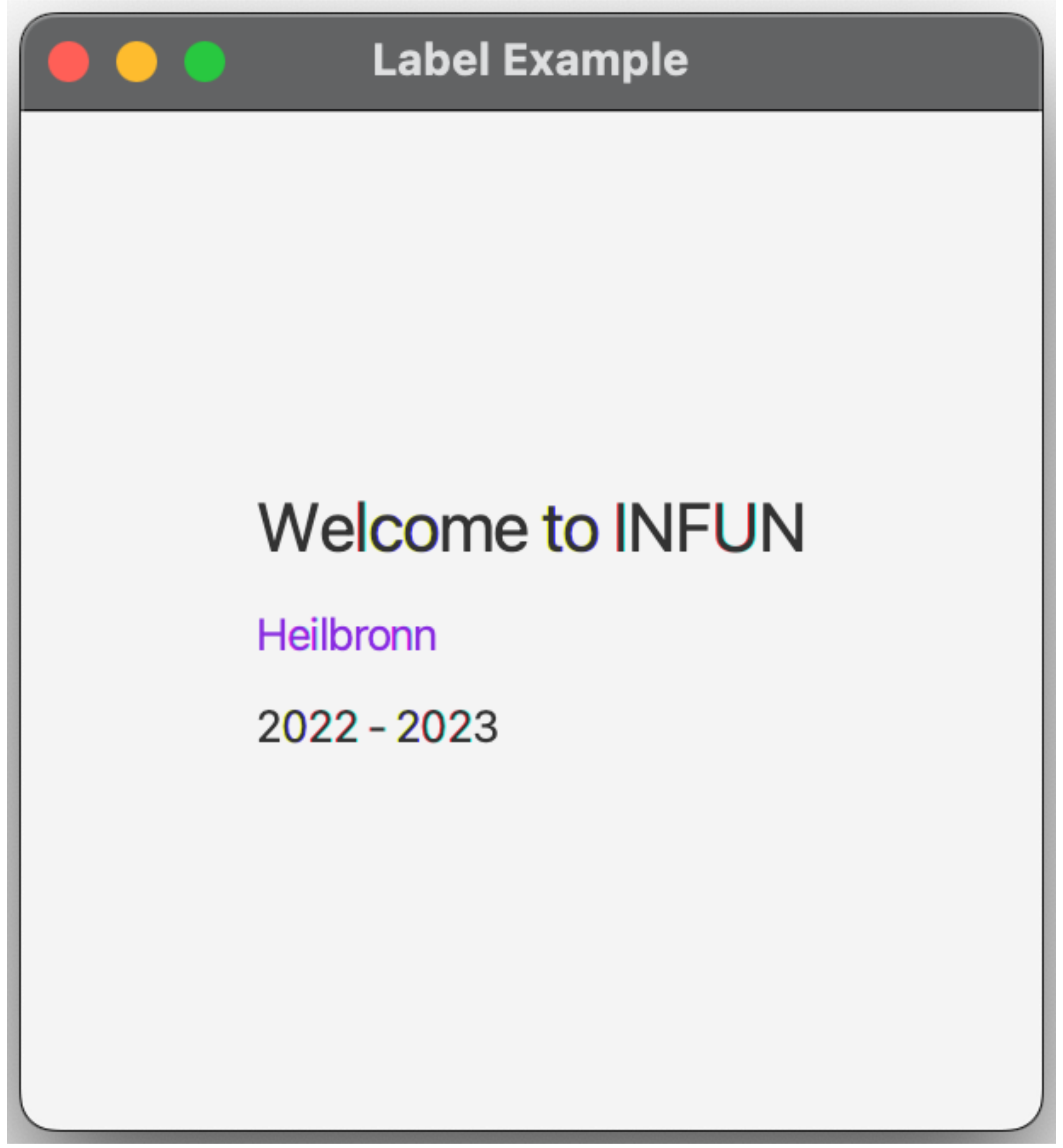
- Label
- Button
- Radio Button
- Toggle Button
- Checkbox
- Choice Box
- Text Field
- File Chooser
- Color Picker
- Password Field
- Scroll Bar
- Scroll Pane
- List View
- Table View
- Tree View
- Tree Table View
- Combo Box
- Pagination Control
- Separator
- Slider
- Progress Bar and Progress Indicator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

More information on: https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

Label

```
public class LabelApp extends Application {
    @Override
    public void start(Stage stage) {
        GridPane gpane = new GridPane();
        gpane.setAlignment(Pos.CENTER);
        gpane.setVgap(10);
        gpane.setPadding(new Insets(25, 25, 25, 25));
        Label label1 = new Label("Welcome to INFUN");
        label1.setFont(new Font("Cambria", 20));
        gpane.add(label1, 0, 0);
        Label label2 = new Label("Heilbronn");
        label2.setTextFill(Color.BLUEVIOLET);
        gpane.add(label2, 0, 1);
        Label label3 = new Label("2022 - 2023");
        gpane.add(label3, 0, 2);
        Scene scene = new Scene(gpane, 300, 300);
        stage.setTitle("Label Example");
        stage.setScene(scene);
        stage.show();

}
```

Provide several properties to the label



Welcome to INFUN

Heilbronn

2022 - 2023

Button

Buttons fire action events when they are activated
(e.g. clicked, a keybinding for the button is pressed, ...)

```java
Button button = new Button();
button.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});
```

If you are using Java 8+, you can use lambdas for action listeners

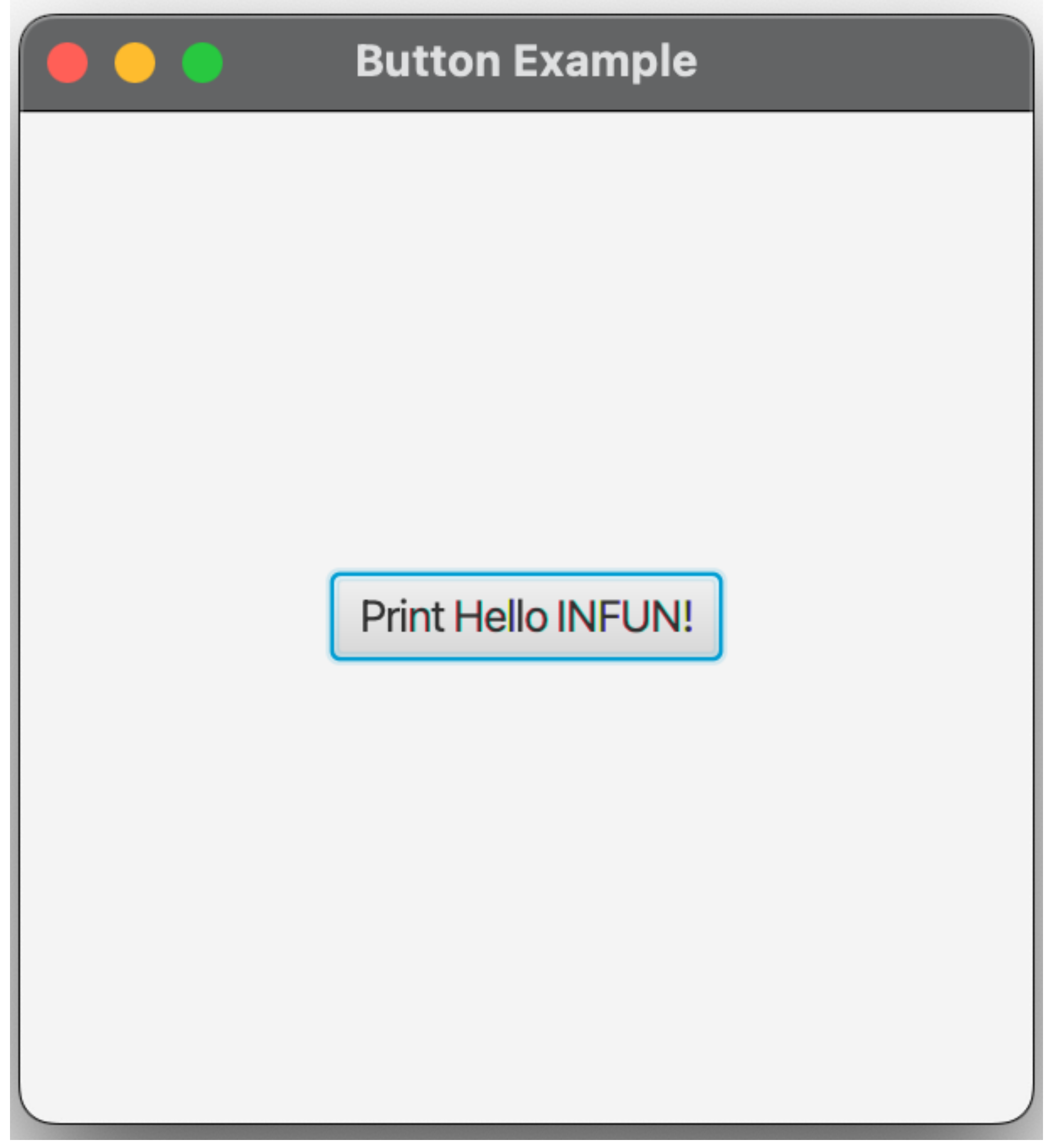
```java
Button button = new Button();
button.setOnAction((ActionEvent action) -> System.out.println("Hello World!"));
// or
button.setOnAction(action -> System.out.println("Hello World!"));
```

Button

```java
public class ButtonApp extends Application {
    @Override
    public void start(Stage stage) {
        VBox vBox = new VBox();
        vBox.setAlignment(Pos.CENTER);
        vBox.setSpacing(10);
        Button button = new Button("Print Hello INFUN!");

        button.setOnAction(action -> System.out.println("Hello INFUN!"));

        vBox.getChildren().addAll(button);
        Scene scene = new Scene(vBox, 300, 300);
        stage.setTitle("Button Example");
        stage.setScene(scene);
        stage.show();
    }
}
```

You can provide some properties to the button

When the user presses the button, the lambda function is invoked



Output

```
Hello INFUN!
```

Button

Buttons can have a **`graphic`** element: this can be any JavaFX node, like a **`ProgressBar`**

```
button.setGraphic(new ProgressBar(-1));
```

An **`ImageView`**

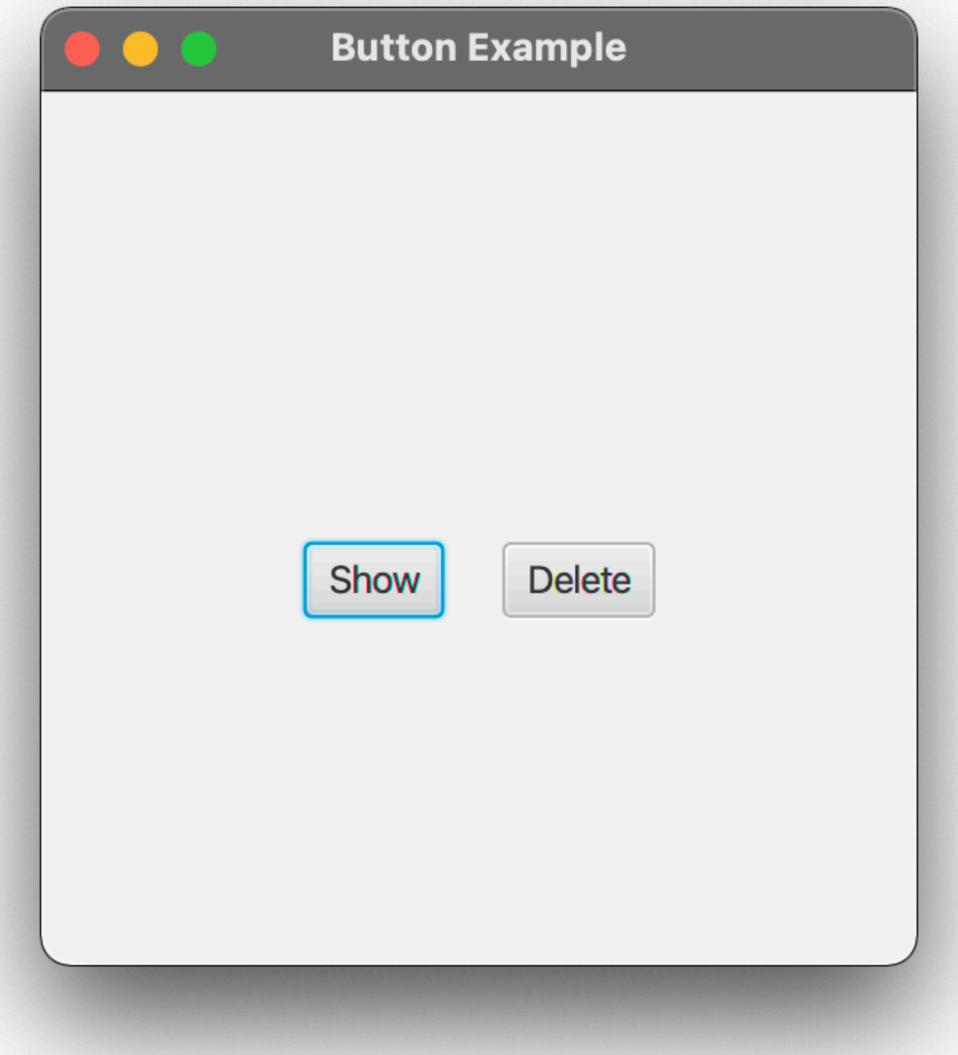
```
button.setGraphic(new ImageView("images/icon.png"));
```

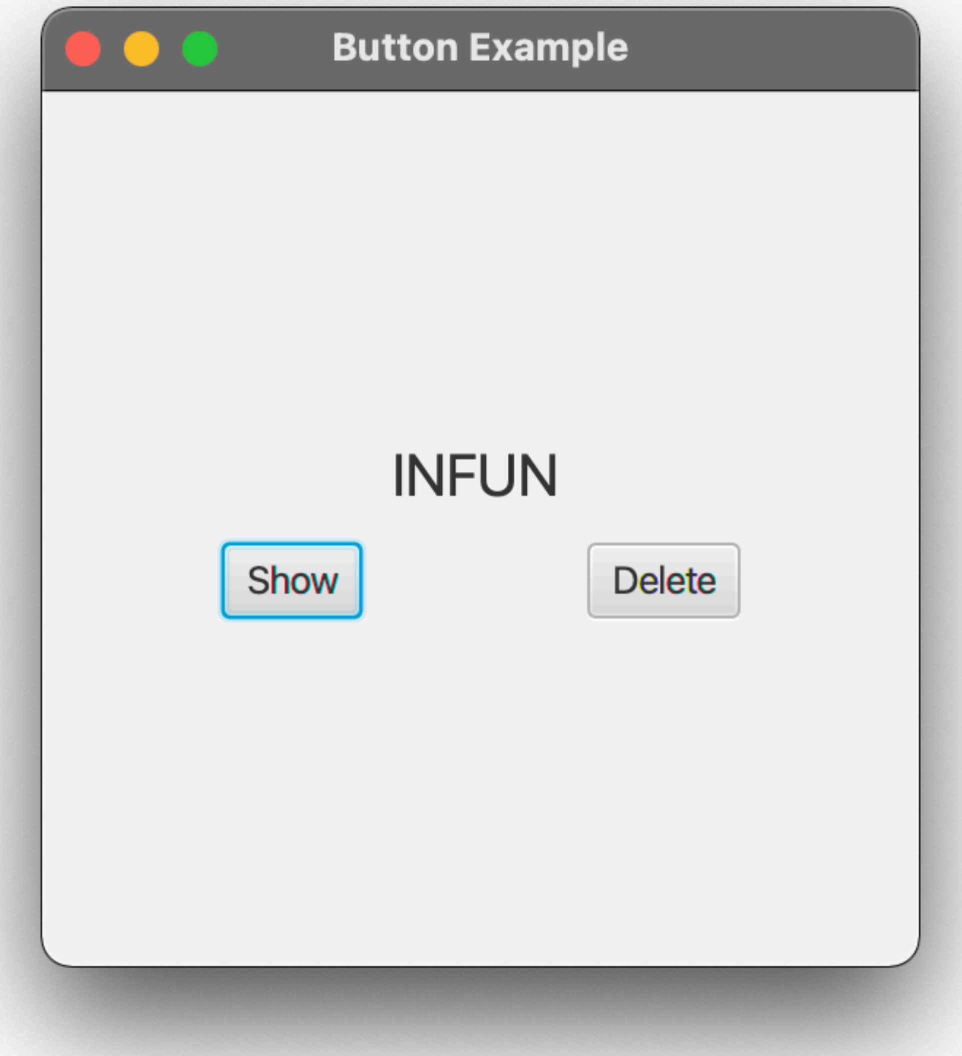
Or even another button

```
button.setGraphic(new Button("Test"));
```

Button

```
public class ButtonApp extends Application {
    @Override
    public void start(Stage stage) {
        GridPane gpane = new GridPane();
        gpane.setAlignment(Pos.CENTER);
        gpane.setHgap(10);
        gpane.setVgap(10);
        gpane.setPadding(new Insets(25, 25, 25, 25));

        Label label1 = new Label("");
        label1.setFont(new Font("Cambria", 20));
        gpane.add(label1, 1, 0);

        Button button1 = new Button("Show");
        button1.setOnAction(action -> label1.setText("INFUN"));
        gpane.add(button1, 0, 1);

        Button button2 = new Button("Delete");
        button2.setOnAction(action -> label1.setText(""));
        gpane.add(button2, 2, 1);

        Scene scene = new Scene(gpane, 300, 300);
        stage.setTitle("Button Example");
        stage.setScene(scene);
        stage.show();
    }
}
```

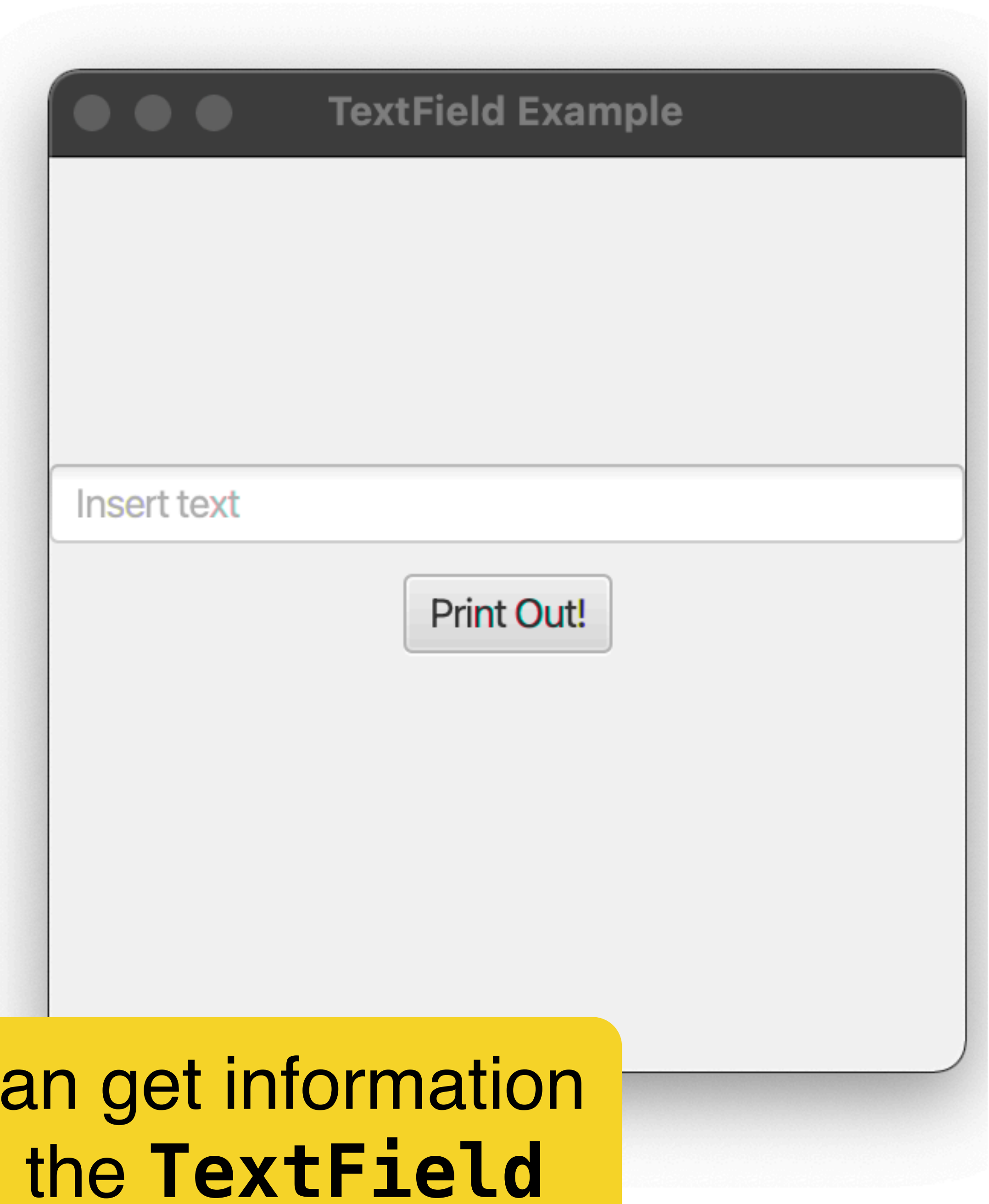


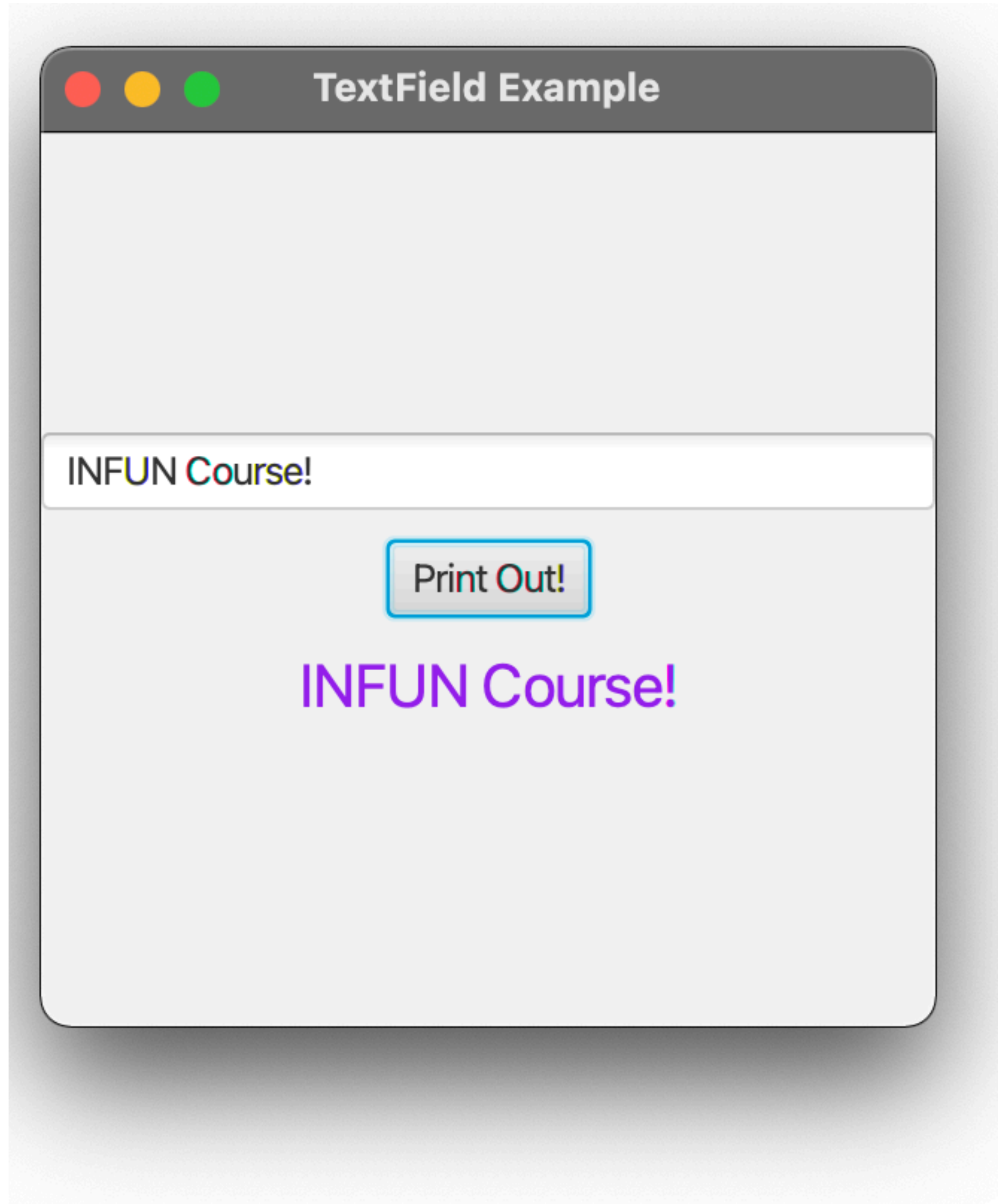


Text field

```
public class TextFieldApp extends Application {
    @Override
    public void start(Stage stage) {
        VBox vBox = new VBox();
        vBox.setAlignment(Pos.CENTER);
        vBox.setSpacing(10);

        TextField txt = new TextField();
        txt.setPromptText("Insert text");

        Label label1 = new Label("");
        label1.setFont(new Font("Cambria", 20));

        Button button1 = new Button("Print Out!");
        button1.setOnAction(action -> {
            label1.setText(txt.getText());
            label1.setTextFill(Color.BLUEVIOLET);
        });

        vBox.getChildren().addAll(txt, button1, label1);
        Scene scene = new Scene(vBox, 300, 300);
        stage.setTitle("TextField Example");
        stage.setScene(scene);
        stage.show();
    }
}
```

You can provide several properties to the **TextField**

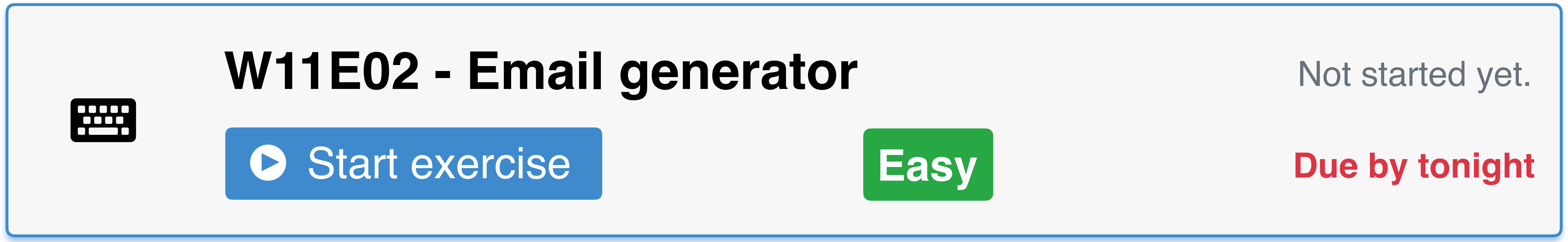




INFUN Course!

INFUN Course!

You can get information from the **TextField**



W11E02 - Email generator

Not started yet.

Due by tonight

10 min

3 pts

- **Problem statement**: develop the user interface of an email generator
  - Users can input their first name and the institution they belong to with two **`TextField`**s
  - The button "Generate" will print out your email in a label in this format

    name@institution.de

  - Hint: you can re-use the code from the previous slide

Outline

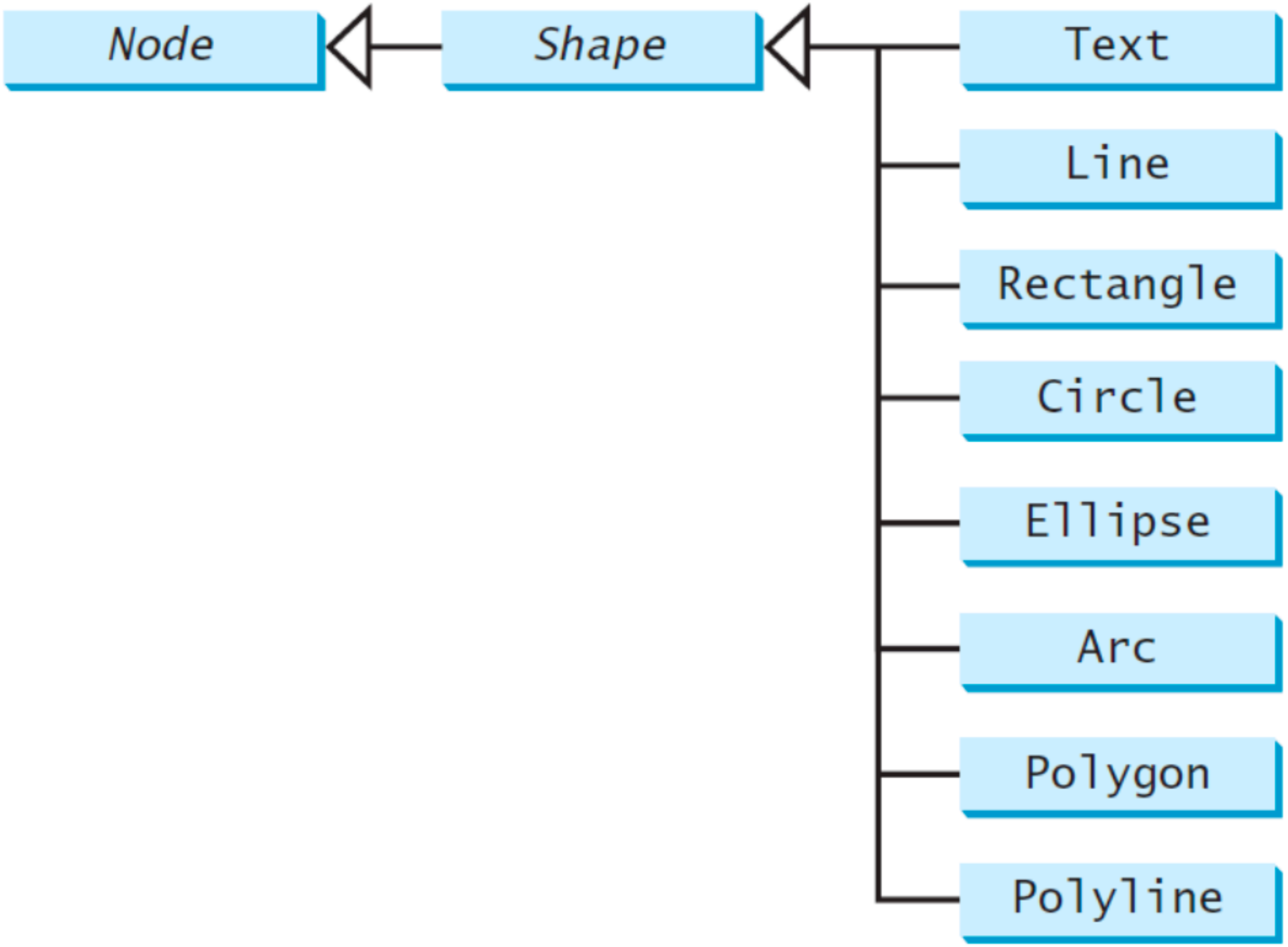
- Usability
- JavaFX
- Layout
- User input

➡ Shapes

- Styling

Shapes

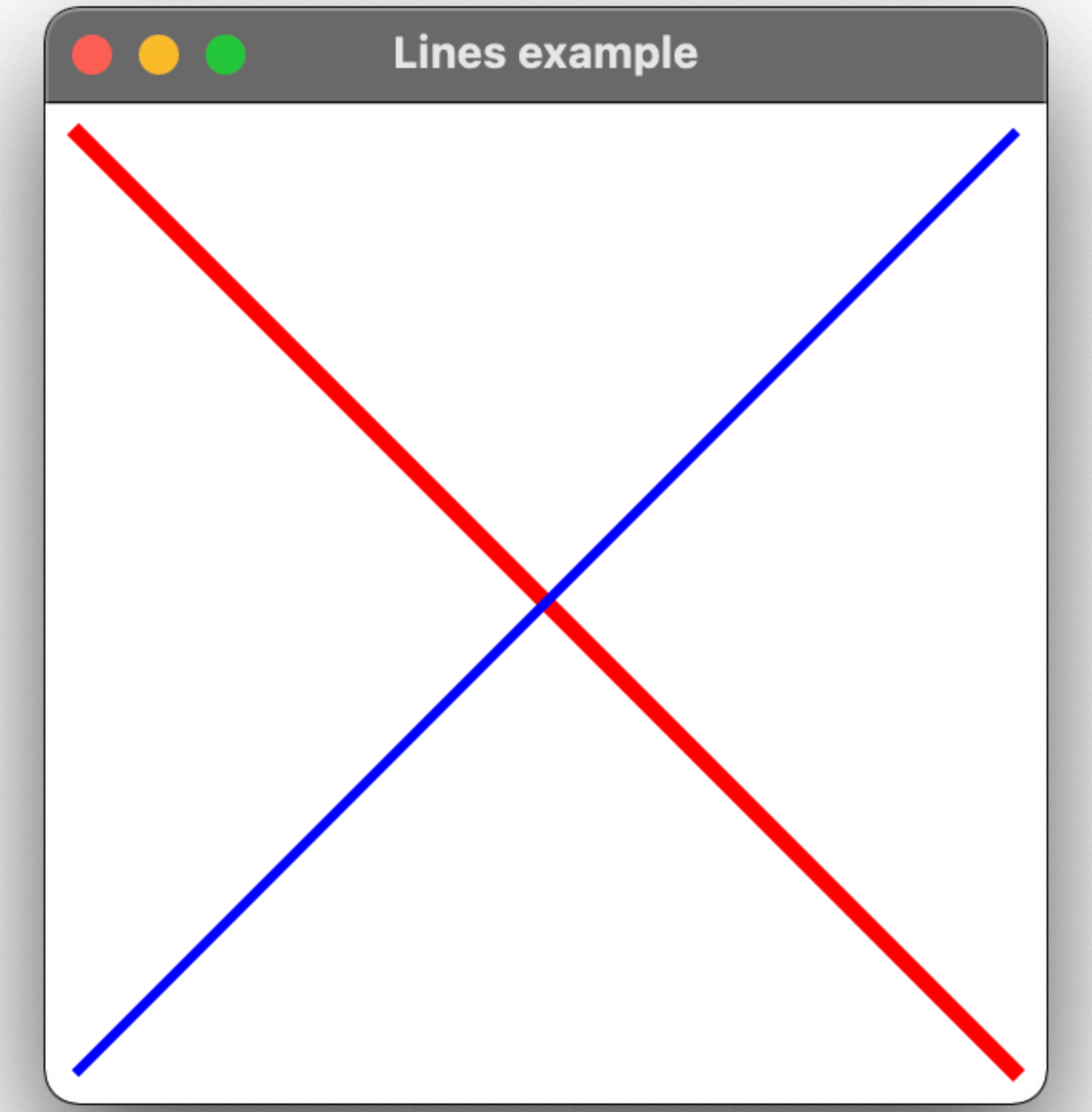
- JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines



Line

```
public class LineApp extends Application {
    @Override
    public void start(Stage stage) {
        Pane layout = new Pane();

        Line line1 = new Line(10, 10, 10, 10);
        line1.endXProperty().bind(layout.widthProperty().subtract(10));
        line1.endYProperty().bind(layout.heightProperty().subtract(10));
        line1.setStrokeWidth(5);
        line1.setStroke(Color.RED);

        Line line2 = new Line(10, 10, 10, 10);
        line2.startXProperty().bind(layout.widthProperty().subtract(10));
        line2.endYProperty().bind(layout.heightProperty().subtract(10));
        line2.setStrokeWidth(3);
        line2.setStroke(Color.BLUE);

        layout.getChildren().add(line1);
        layout.getChildren().add(line2);

        Scene scene = new Scene(layout, 300, 300);
        stage.setTitle("Lines example");
        stage.setScene(scene);
        stage.show();
    }
}
```

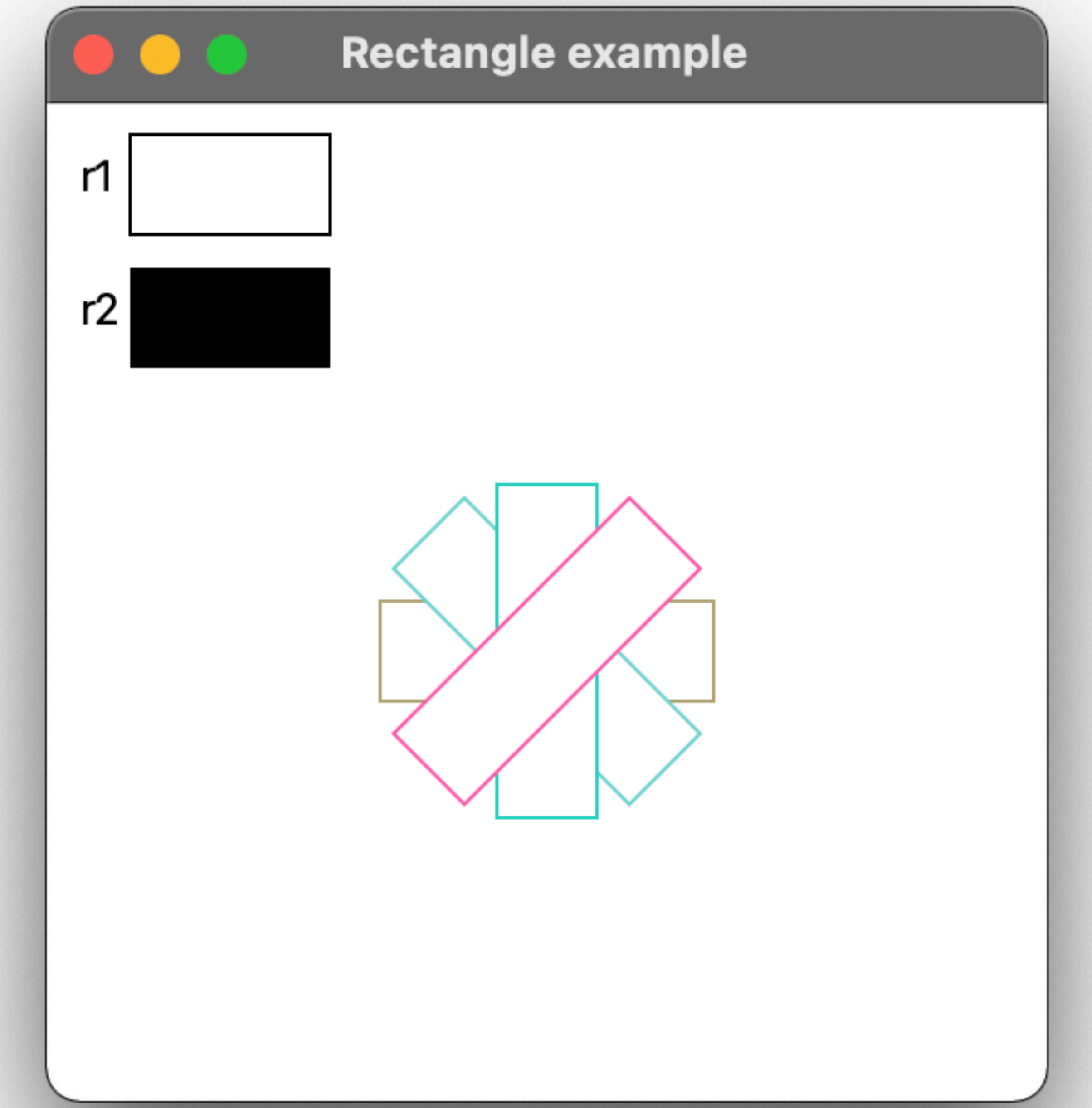
startX, startY, endX, endY



Rectangle

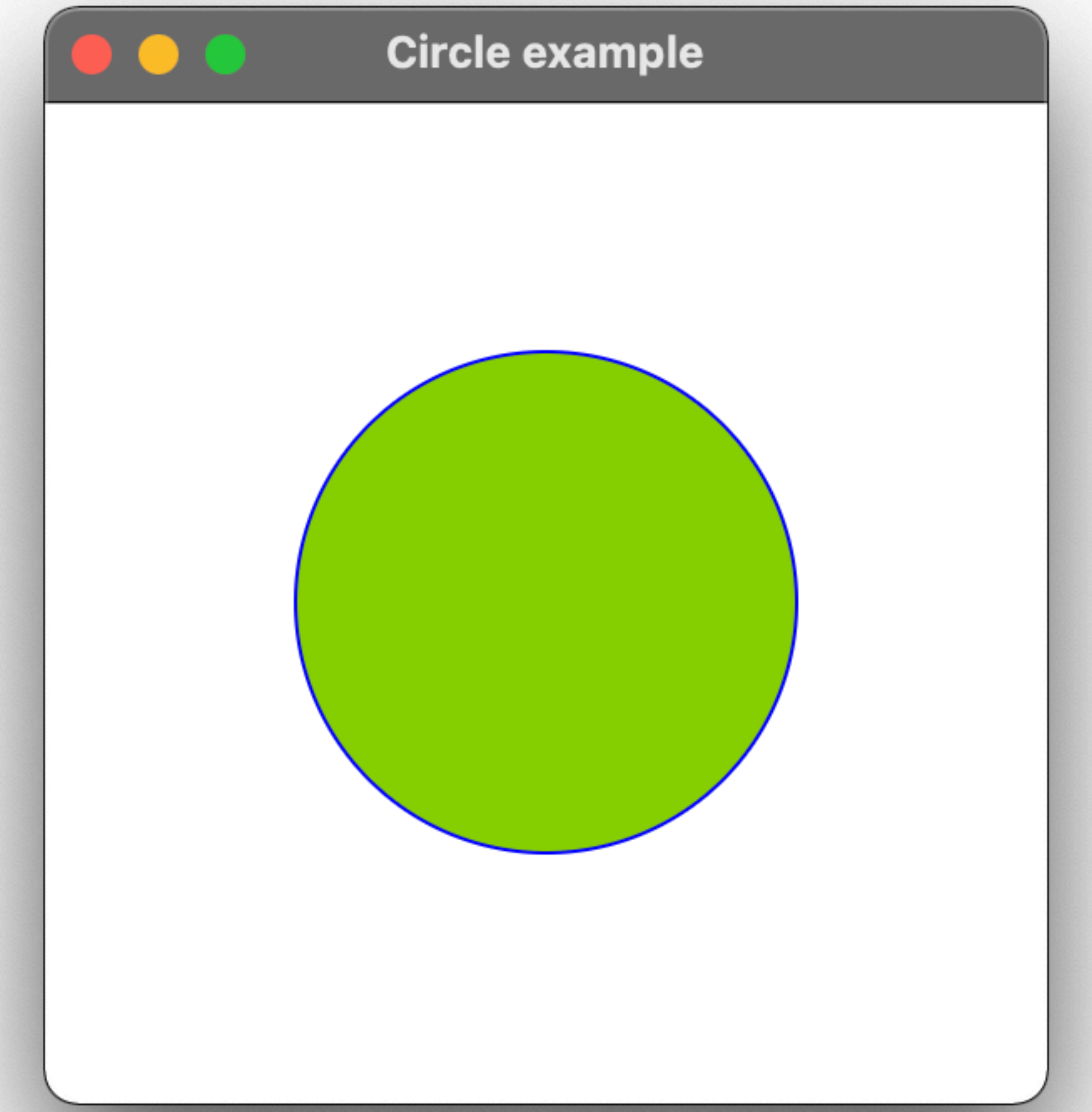
```
public class RectangleApp extends Application {
    @Override
    public void start(Stage stage) {
        Pane layout = new Pane();

        Rectangle rect1 = new Rectangle(25, 10, 60, 30);
        rect1.setStroke(Color.BLACK);
        rect1.setFill(Color.WHITE);
        Rectangle rect2 = new Rectangle(25, 50, 60, 30);

        layout.getChildren().add(new Text(10, 27, "r1"));
        layout.getChildren().add(rect1);
        layout.getChildren().add(new Text(10, 67, "r2"));
        layout.getChildren().add(rect2);

        for (int i = 0; i < 4; i++) {
            Rectangle rect = new Rectangle(150, 75, 100, 30);
            rect.setRotate(i * 360.0 / 8.0);
            Color color = Color.color(Math.random(), Math.random(), Math.random());
            rect.setStroke(color);
            rect.setFill(Color.WHITE);
            layout.getChildren().add(rect);
        }
        Scene scene = new Scene(layout, 300, 300);
        stage.setTitle("Rectangle example");
        stage.setScene(scene);
        stage.show();
    }
}
```

x, y, width, height



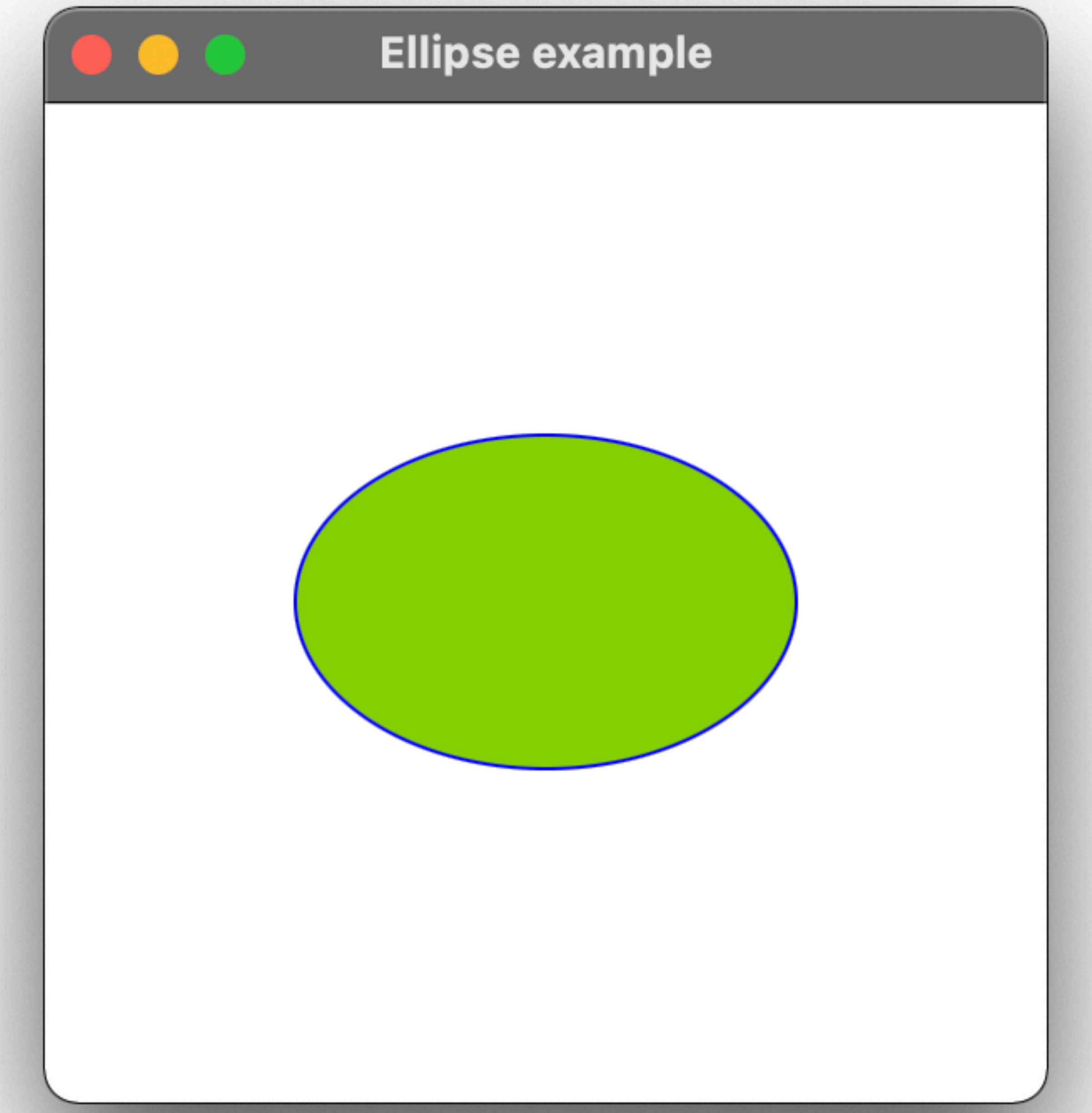
Circle

```
public class CircleApp extends Application {
    @Override
    public void start(Stage stage) {
        Pane layout = new Pane();

        Circle circle = new Circle();
        circle.setCenterX(150);
        circle.setCenterY(150);
        circle.setRadius(75);
        circle.setStroke(Color.BLUE);
        circle.setFill(Color.YELLOWGREEN);

        layout.getChildren().add(circle);
        Scene scene = new Scene(layout, 300, 300);
        stage.setTitle("Circle example");
        stage.setScene(scene);
        stage.show();
    }
}
```



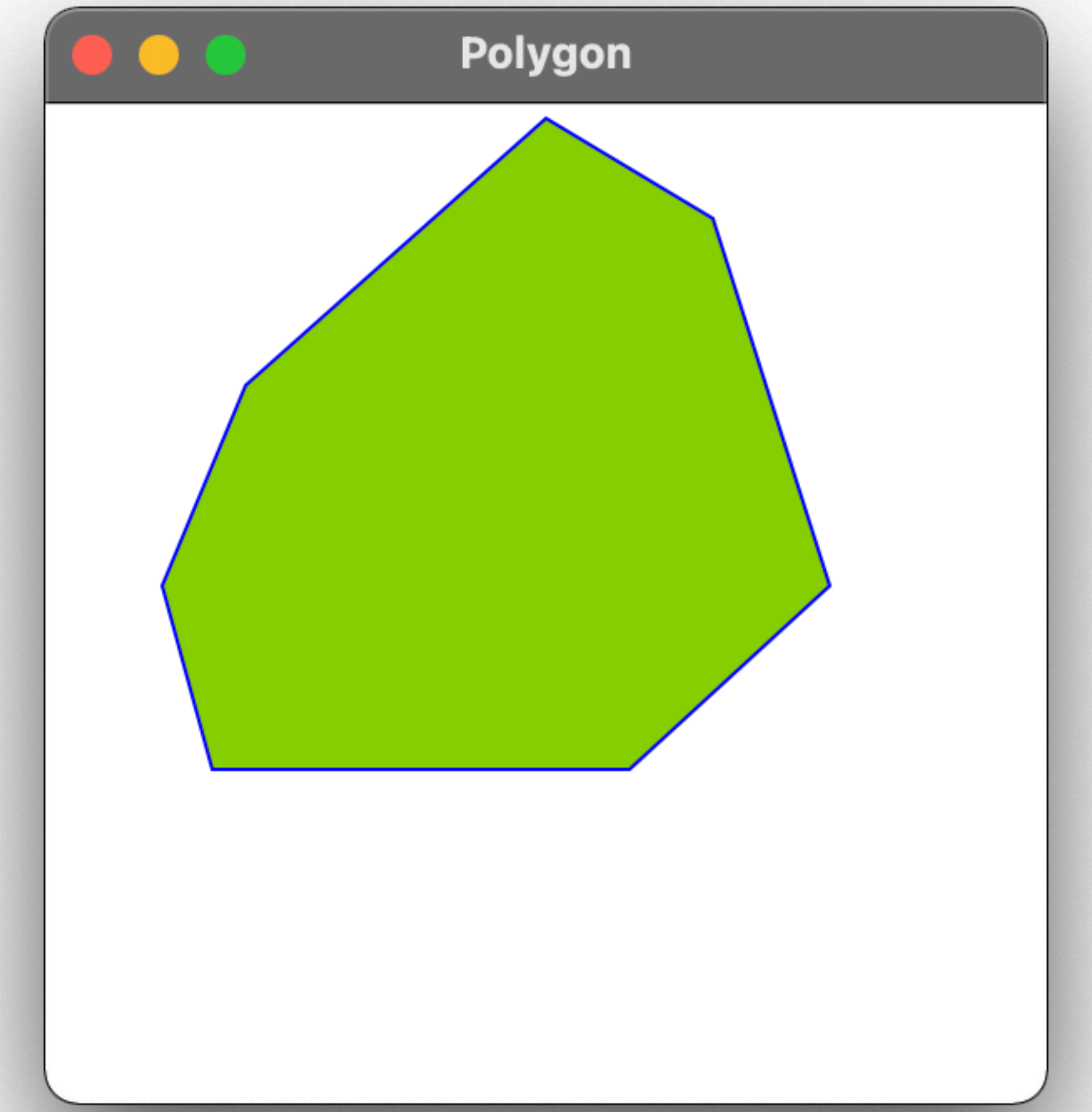
Ellipse

```
public class EllipseApp extends Application {
    @Override
    public void start(Stage stage) {
        Pane layout = new Pane();

        Ellipse ellipse = new Ellipse();
        ellipse.setCenterX(150);
        ellipse.setCenterY(150);
        ellipse.setRadiusX(75);
        ellipse.setRadiusY(50);
        ellipse.setStroke(Color.BLUE);
        ellipse.setFill(Color.YELLOWGREEN);

        layout.getChildren().add(ellipse);
        Scene scene = new Scene(layout, 300, 300);
        stage.setTitle("Ellipse example");
        stage.setScene(scene);
        stage.show();
    }
}
```



Polygon

```
public class PolygonApp extends Application {
    @Override
    public void start(Stage stage) {
        Pane layout = new Pane();

        Polygon polygon = new Polygon();
        polygon.getPoints().addAll(150.0, 5.0, 200.0,
              35.0, 235.0, 145.0, 175.0, 200.0, 50.0,
              200.0, 35.0, 145.0, 60.0, 85.0);

        polygon.setStroke(Color.BLUE);
        polygon.setFill(Color.YELLOWGREEN);

        layout.getChildren().add(polygon);

        Scene scene = new Scene(layout, 300, 300);
        stage.setTitle("Polygon example");
        stage.setScene(scene);
        stage.show();
    }
}
```



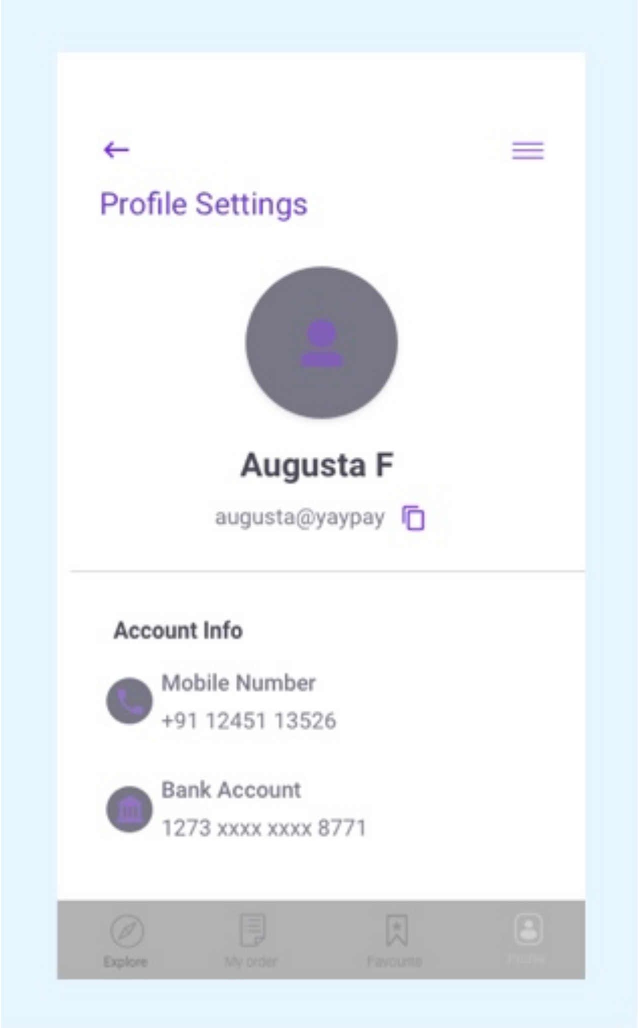
Outline

- Usability
- JavaFX
- Layout
- User input
- Shapes

➡ Styling

Styling

- JavaFX allows to apply style properties to **`Stage`**s, **`Layout`**s, and **`Control`**s
- Providing style to the GUI will improve the **user experience** of the program



←

Profile Settings

Augusta F

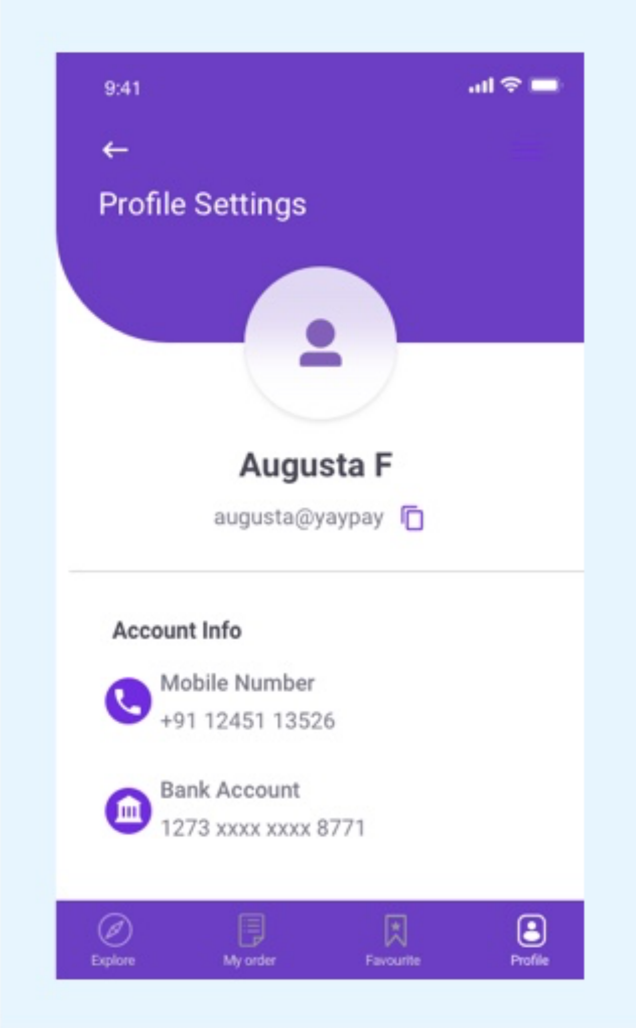
augusta@yaypay

Account Info

Mobile Number
+91 12451 13526

Bank Account
1273 xxxx xxxx 8771

✗



9:41

←

Profile Settings

Augusta F

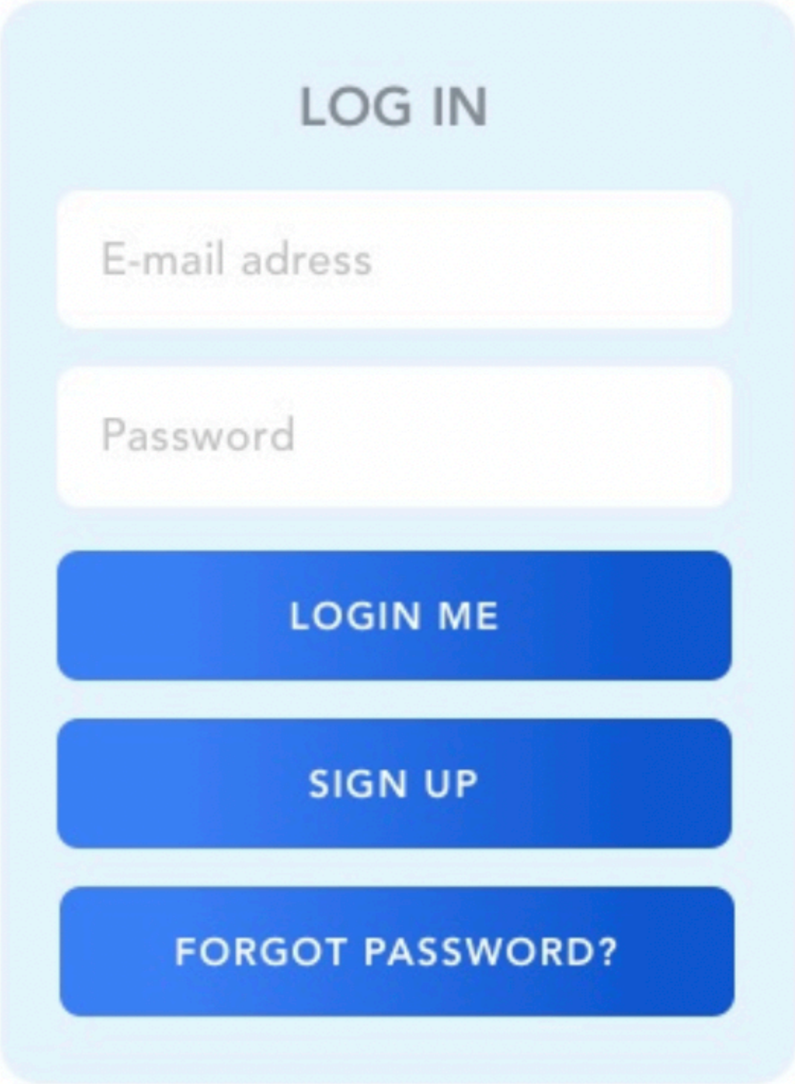
augusta@yaypay

Account Info

Mobile Number
+91 12451 13526

Bank Account
1273 xxxx xxxx 8771

✓

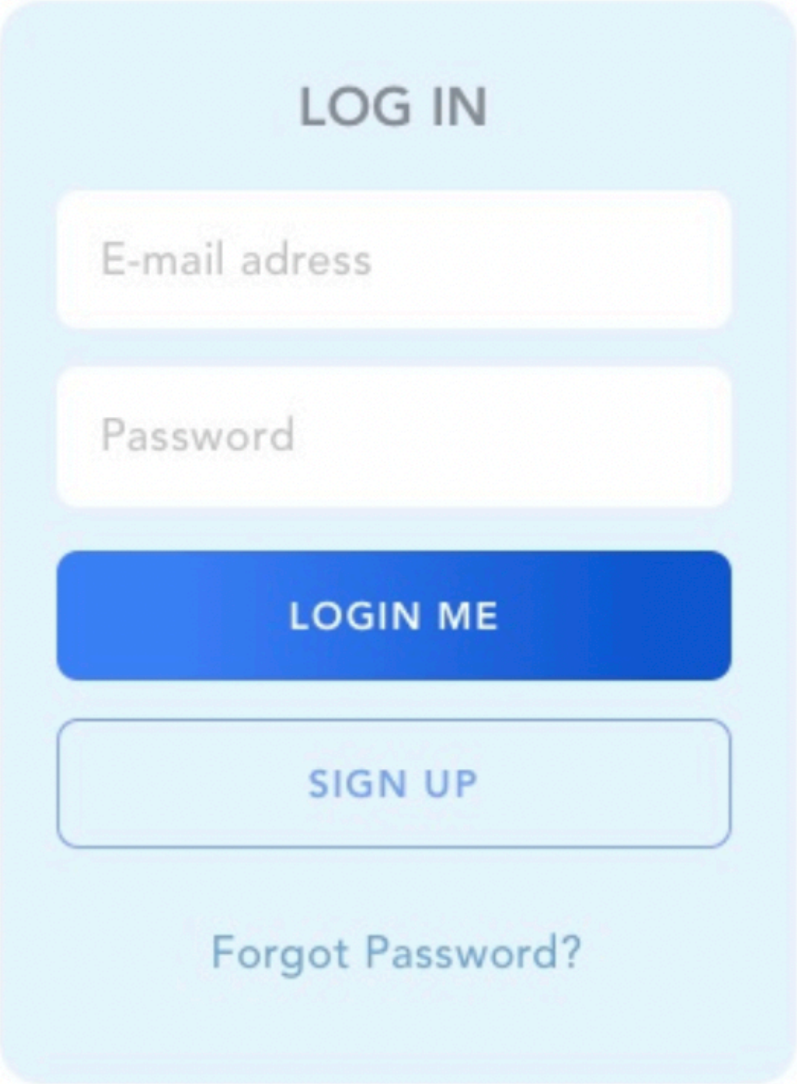


LOG IN

E-mail adress

Password

✗



LOG IN

E-mail adress

Password

✓

Font

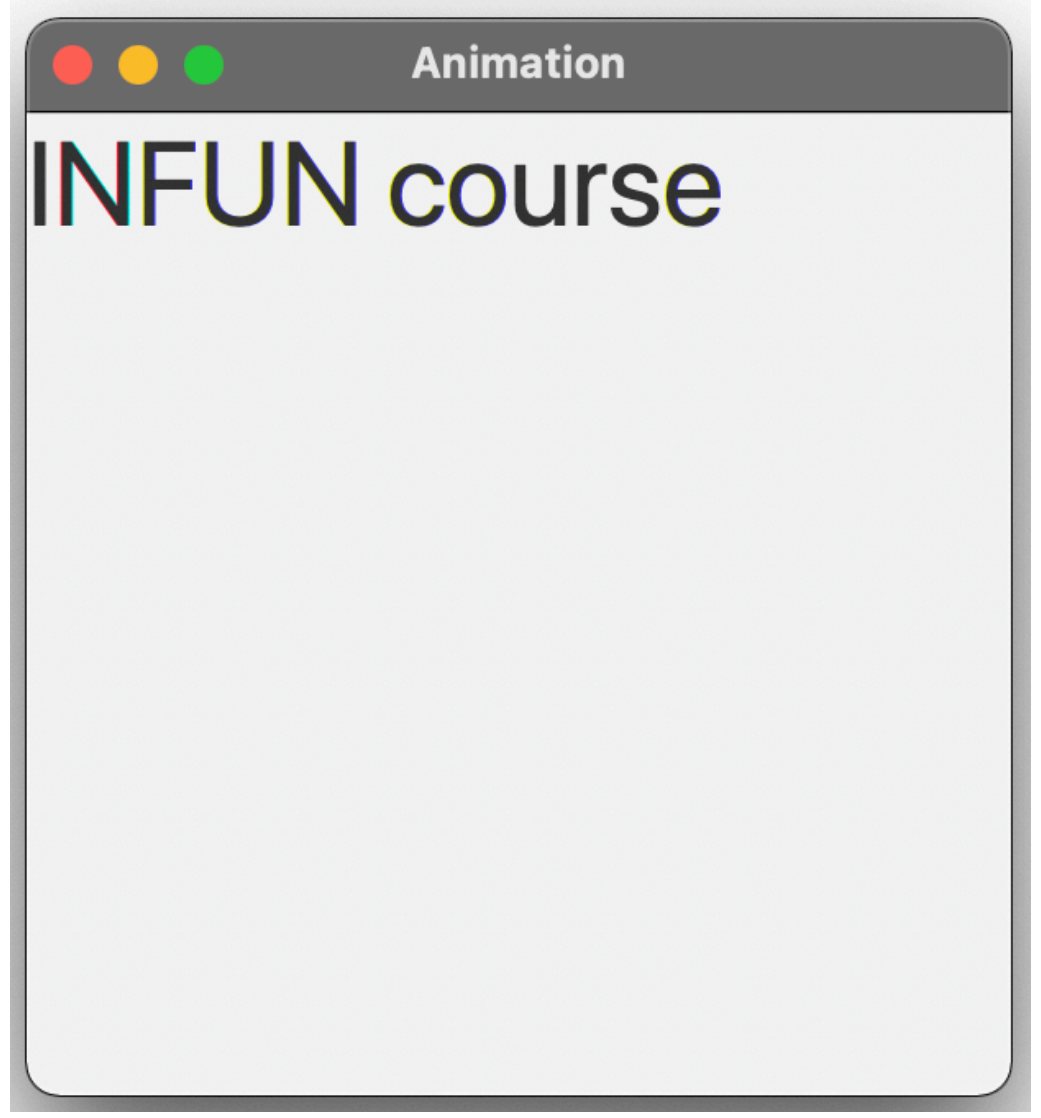
You can set the font of a JavaFX control using the **`setFont()`** method

```
label.setFont(Font.font("Calibri"));
```

`javafx.scene.text.Font` is used in this example

```
label.setFont(Font.font("Calibri", FontWeight.BOLD, 36));
```

The **`Font`** class also lets you specify the font weight and the font size



INFUN course

Fill color

- You can set the fill color of a control
- The fill color is the "inside" color used to draw the text
- You set the fill color of a control via its **`setTextFill()`** method, which takes a **`Color`** object as a parameter
- The **`Color`** class also has a set of static factory methods that can help you create **`Color`** instances using a variety of different parameters

```
label.setTextFill(Color.GREEN);
```

```
label.setTextFill(Color.web("#ffc0cb"));   //Pink
```

Creates a **`Color`** instance based on a traditional **web color code**

```
label.setTextFill(Color.rgb(100, 200, 0)); //Green
```

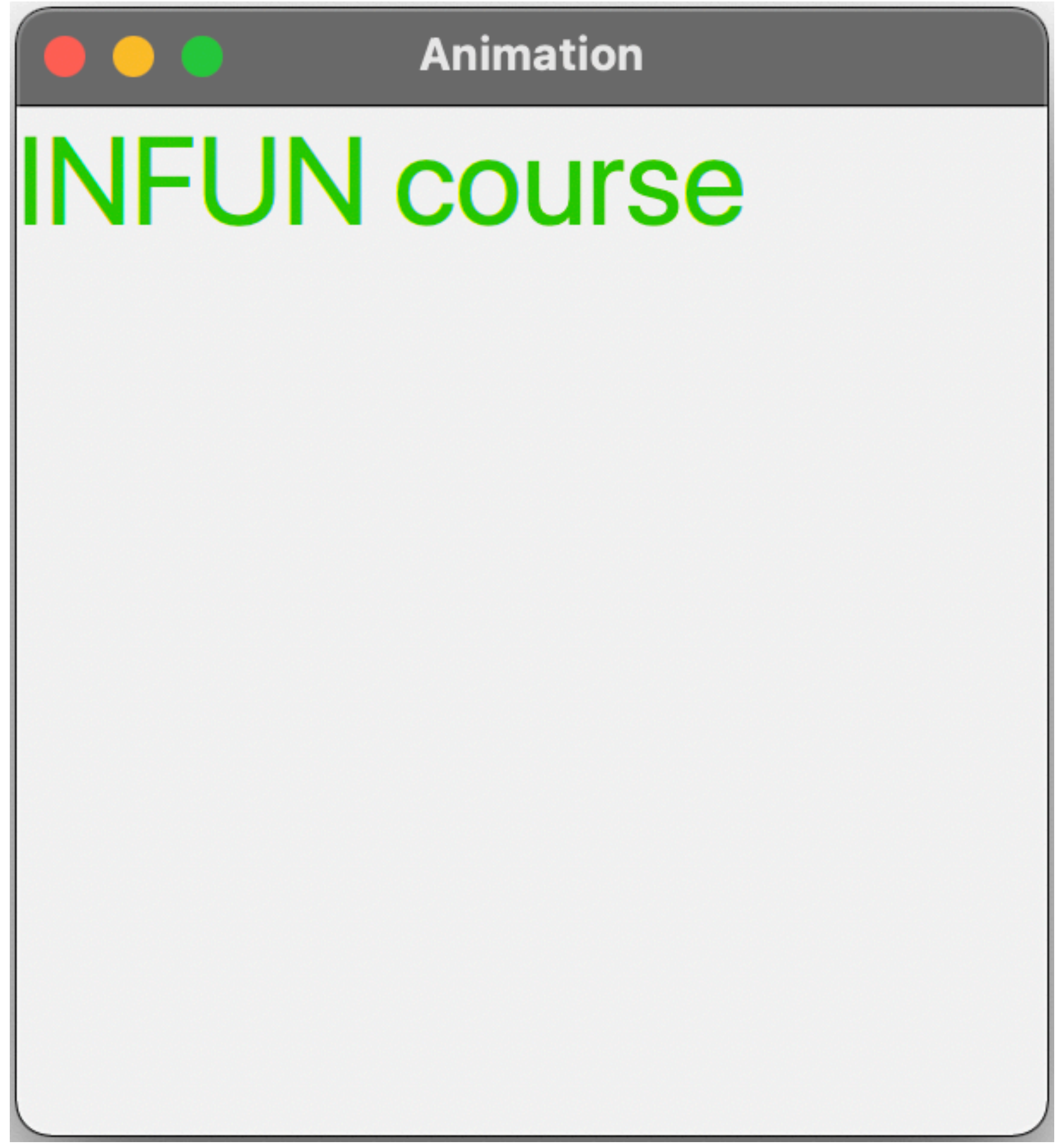
Creates a **`Color`** instance from **red**, **green**, and **blue** color values

```
label.setTextFill(Color.grayRgb(100));   //Gray
```

Creates a **`Color`** instance representing a **gray** color

```
label.setTextFill(Color.hsb(1.0, 0.7, 0.4));   //Brown
```

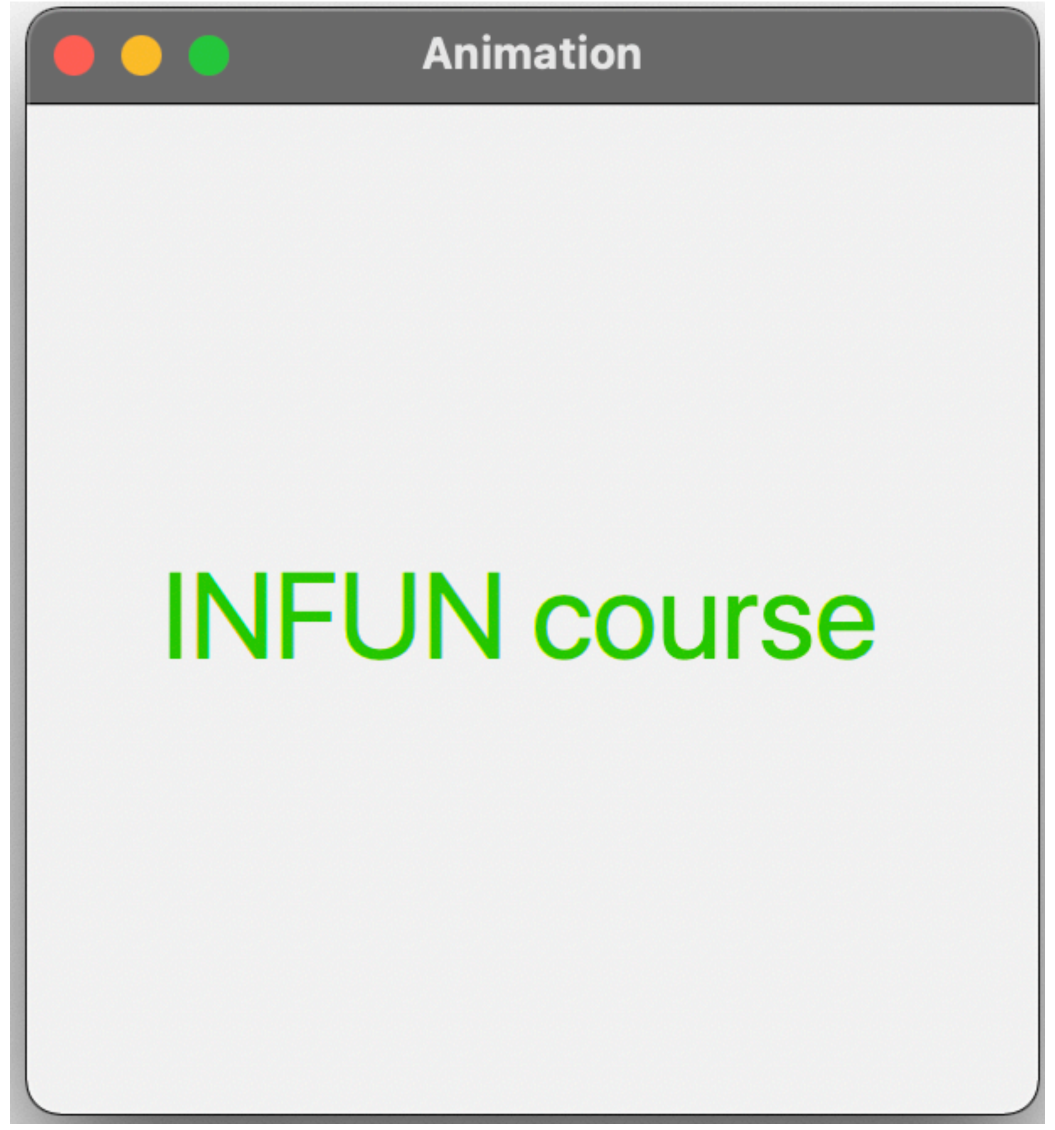
Creates a **`Color`** instance based on **Hue**, **Saturation** and **Brightness** (HSB)



Position

- The X and Y position of a control determines where inside its parent container element the control is displayed - provided the parent container respects this position (Pane does, VBox does not)
- You can set the X and Y position of a control using its methods **`setLayoutX()`** and **`setLayoutY()`**

```
label.setLayoutX(40);
label.setLayoutY(130);
```



INFUN course

CSS styling

- JavaFX enables you to style the components using **CSS**, just like you can style HTML and SVG elements in web pages with CSS
- JavaFX uses the same **CSS** syntax as for the web, but the properties are specific and have slightly different names than their web counterparts
- Styling JavaFX applications using **CSS** helps to **separate** styling (looks) from the application code
- This results in a **cleaner application code** and makes it easier to change the styling of the application or to support multiple themes (e.g., light vs. dark)

CSS styling

- CSS: **c**ascading **s**tyle **s**heets
- Simple (domain specific) language that specifies how a user interface appears
- Originally created for the web
- You can use CSS to style a **JavaFX user interface**
- A style sheet is a text file containing one or more **style definitions**, written in the following general format

```
selector {
    property: value;
    property: value;
}
```

CSS styling example

- This style definition specifies that label controls should display their text in a cursive, 14-point, italic, bold font, with a dotted border around the control

```
.label {
    -fx-font-family: cursive;
    -fx-font-size: 14pt;
    -fx-font-style: italic;
    -fx-font-weight: bold;
    -fx-border-style: dotted;
}
```

W11E03 - TUM Logo

Not started yet.

Due by tonight

10 min

4 pts

- **Problem statement**: create the **TUM Logo** in JavaFX using rectangles
- **Hint:** you can use the following code

```
public class TUMLogoApplication extends Application {
    private static final Color TUM_BLUE = Color.rgb(0, 101, 189); // TUM Blue
    private static final double UNIT = 50; // One measurement unit
    private final Pane pane = new Pane();

    @Override
    public void start(Stage stage) {

        // ... Add rectangles using the method below to draw the TUM Logo
        Scene scene = new Scene(pane, 24 * UNIT, 15 * UNIT);
        stage.setTitle("TUM Logo");
        stage.setScene(scene);
        stage.show();
    }
    private void addRectangleToPane(double x, double y, double width, double height, Color color) {
        Rectangle rectangle = new Rectangle(x, y, width, height);
        rectangle.setFill(color);
        pane.getChildren().add(rectangle);
    }
}
```

Next steps

- **Tutor group** exercise
  - T11E01 - Number Conversion
- **Homework** exercise
  - H11E01 - Welcome to SealTemis
- **Project work**
  - Implement the game
- Read the following articles
  - https://www.baeldung.com/javafx
  - https://www.vojtechruzicka.com/javafx-getting-started

→ Due by **Wednesday, January 14, 13:00**

Summary

- **Usability** and **user experience** are important aspects in programming and software engineering
  - They can be a deciding factor whether your application is successful or not
  - **Prototyping** allows to experiment quickly and to identify strengths and weaknesses of the designed graphical user interface (GUI)
- There are different GUI frameworks for different platforms and programming languages
- They all have common characteristics such as **layouts**, **controls**, **shapes**, **styling**
- **JavaFX** is **one example** of a GUI framework for Java-based applications

References

- J. Nielsen, Usability Engineering, Academic Press, 1993
- Recommendation: D. Norman, The Design of Everyday Things, Doubleday, 1998
- https://www.nngroup.com/articles/ten-usability-heuristics
- J. Nielsen, How to conduct a Heuristic Evaluation https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation
- H. Petroski, Success through Failure: The Paradox of Design, Princeton Press, 2008
- Recommendation: The Iceberg Secret Revealed https://www.joelonsoftware.com/articles/fog0000000356.html
- P. M. Fitts, The information capacity of the human motor system in controlling the amplitude of movement. Journal of Experimental Psychology, 47, 381-391, 1954
- K. Popper, Objective Knowledge: An Evolutionary Approach, Oxford University, 1972
- https://www.baeldung.com/javafx
- https://www.vojtechruzicka.com/javafx-getting-started

Further readings: user interface design guidelines

- macOS and iOS user experience
https://developer.apple.com/design/human-interface-guidelines/
- Android: https://developer.android.com/design/index.html
- Windows user experience interaction guidelines
https://docs.microsoft.com/en-us/windows/apps/design/