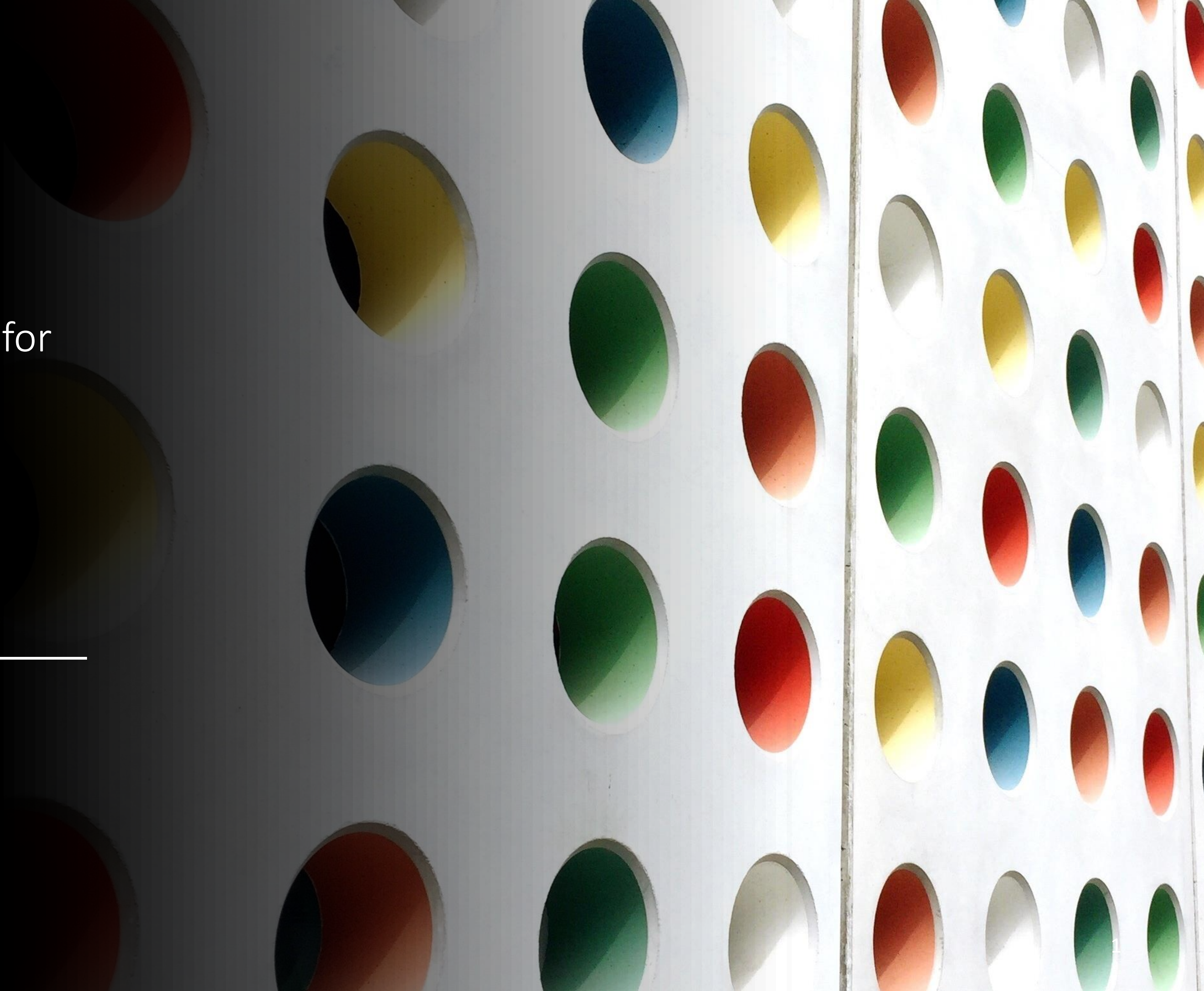


COMP8710 Advanced Java for Programmers

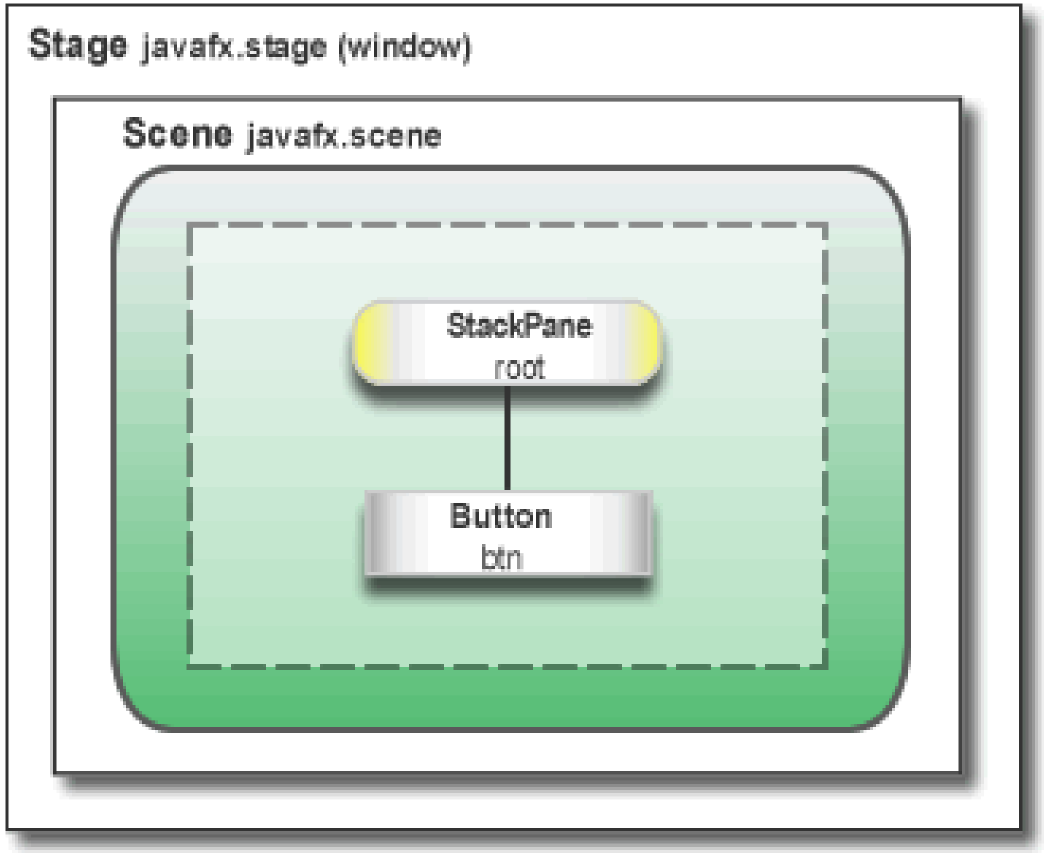
Lecture 16
More JavaFX (1)

Yang He

Topics

- JavaFX Introduction
- JavaFX in IntelliJ IDEA
- Introduction to Apache Maven
- Interacting with users

The theatre metaphor



- Stage: window
- Scene: window content
- (Stack) Pane: layout manager
- Button: UI controls

JavaFX applications: basics

- The main class for a JavaFX application
  - extends `javafx.application.Application`
  - overrides the `start` method which is automatically called when the application is launched, i.e. calling the method `launch`, from within the main method
- Note:
  - A `Stage` object is essentially a window
  - A primary Stage is automatically created by the JVM when the application is launched
  - You can create additional Stage objects if you want to open additional windows

Introduction to Apache Maven

- Maven is a powerful project management tool
- It manages
  - Project build
  - Dependencies + versions
    - External libraries that a project uses
  - Documentation
- Core configuration file: `pom.xml`

Apache Maven Project
http://maven.apache.org/

Why use Maven?

- Key features of Maven
  - Simple project setup
  - Dependency management
    - Dependencies are automatically downloaded/updated from a central repository
- Central repository for dependencies
  - The default central repository is at https://mvnrepository.com/

The default local repository is in the .m2/repository folder under the user's home directory.

Project Object Model (POM)

- Maven configuration in pom.xml
  - Describes the project
  - Manages dependencies
  - Configures plugins for building the software
- Each dependency/plugin is defined by:
  - groupID – Organisation
  - artifactID – Name of the artifact
  - version – Version of the artifact

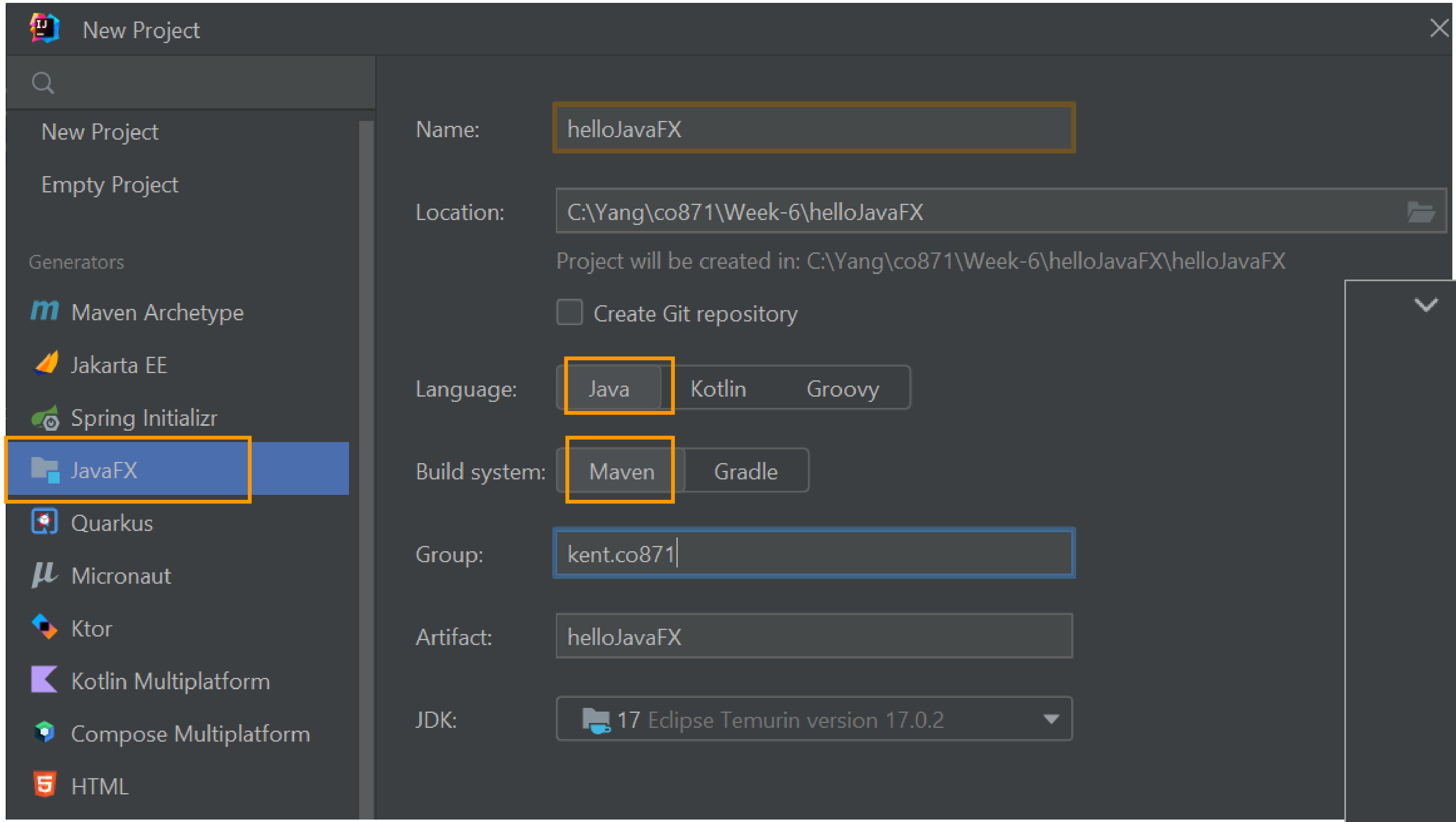
```
<dependency>
   <groupId>org.openjfx</groupId>
   <artifactId>javafx-controls</artifactId>
   <version>21</version>
</dependency>
```

Maven Build Lifecycles

- They include
  - Validate – Checks the correctness
  - Compile – Compiles the source code
  - Test – Run unit tests
  - Package – Packages compiled code into an archive file
  - Install – Installs the package file into the local Maven repository
  - Deploy – Deploys the package file to a remote server or repository

Demo:
Create a JavaFX project in IntelliJ IDEA

`helloJavaFX`


New Project

New Project
Empty Project

Generators
Maven Archetype
Jakarta EE
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform
Compose Multiplatform
HTML

Name: helloJavaFX

Location: C:\Yang\co871\Week-6\helloJavaFX

Project will be created in: C:\Yang\co871\Week-6\helloJavaFX\helloJavaFX

Create Git repository

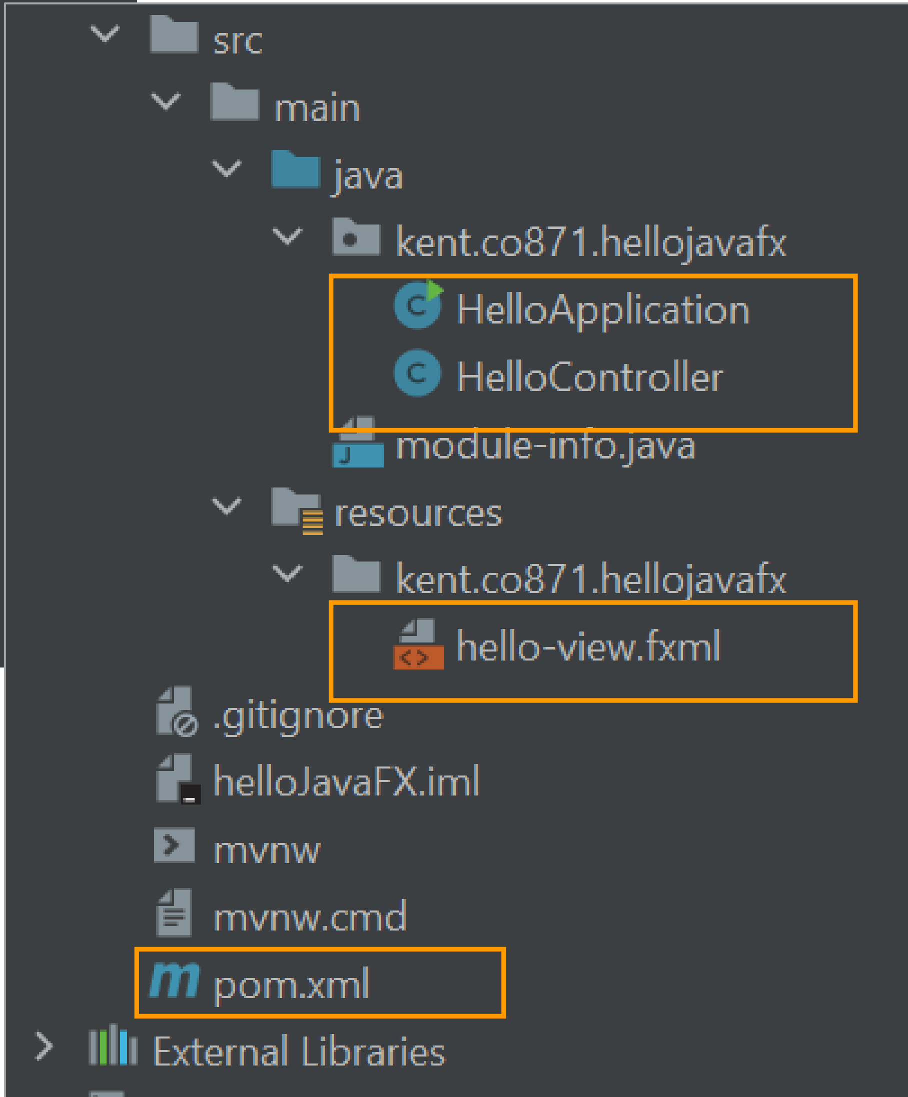
Language: [✓ Java] Kotlin Groovy

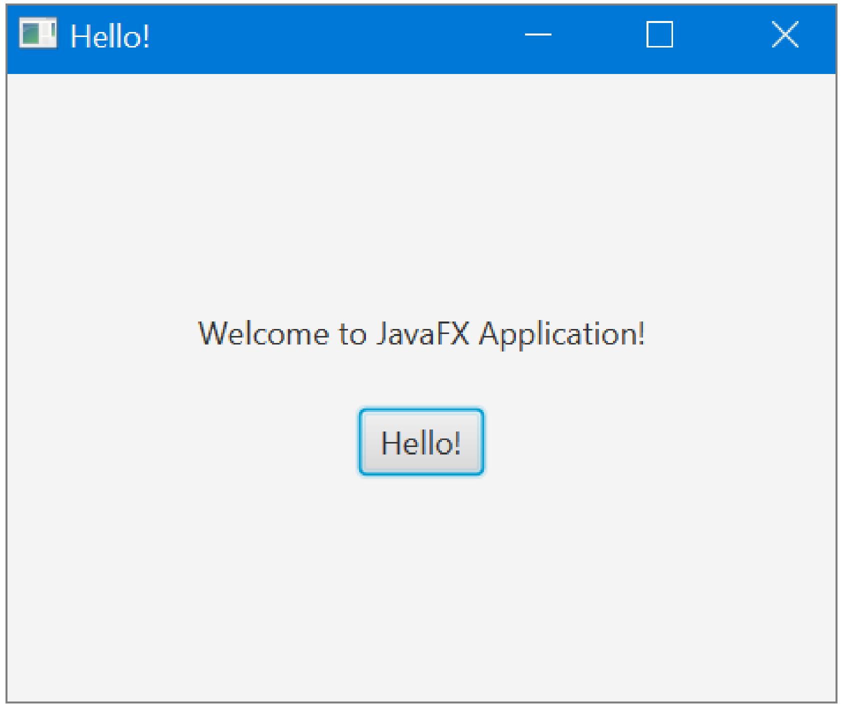
Build system: [✓ Maven] Gradle

Group: kent.co871

Artifact: helloJavaFX

JDK: 17 Eclipse Temurin version 17.0.2


```
src
  main
    java
      kent.co871.hellojavafx
        HelloApplication
        HelloController
      module-info.java
    resources
      kent.co871.hellojavafx
        hello-view.fxml
.gitignore
helloJavaFX.iml
mvnw
mvnw.cmd
pom.xml
External Libraries
```



Welcome to JavaFX Application!

Demo: `HelloApplication.java`

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(
                                HelloApplication.class.getResource("hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }

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

FXMLLoader is a class that loads an .fxml file and creates the UI elements defined in it.

Demo: `hello-view.fxml`

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>

<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
     fx:controller="kent.co871.hellojavafx.HelloController">
   <padding>
      <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
   </padding>

   <Label fx:id="welcomeText"/>
   <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```

Demo: HelloControler.java

```
package kent.co871.hellojavafx;

import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class HelloController {
   @FXML
   private Label welcomeText;

   @FXML
   protected void onHelloButtonClick() {
      welcomeText.setText("Welcome to JavaFX Application!");
   }
}
```

- *We use @FXML to make a field/method accessible to the FXMLLoader.*
- *FXMLLoader can inject the references to the UI elements from the .fxml file into the controller class.*

FXML

- FXML is an XML-based language that provides the structure for building a user interface *separate* from the application logic of your code
  - To load up a `.fxml` file, use `FXMLLoader.load`
  - See this tutorial for more information
- You can use the graphical Scene Builder to generate your FXML
  - See how to Configure JavaFX Scene Builder

JavaFX Application Structure

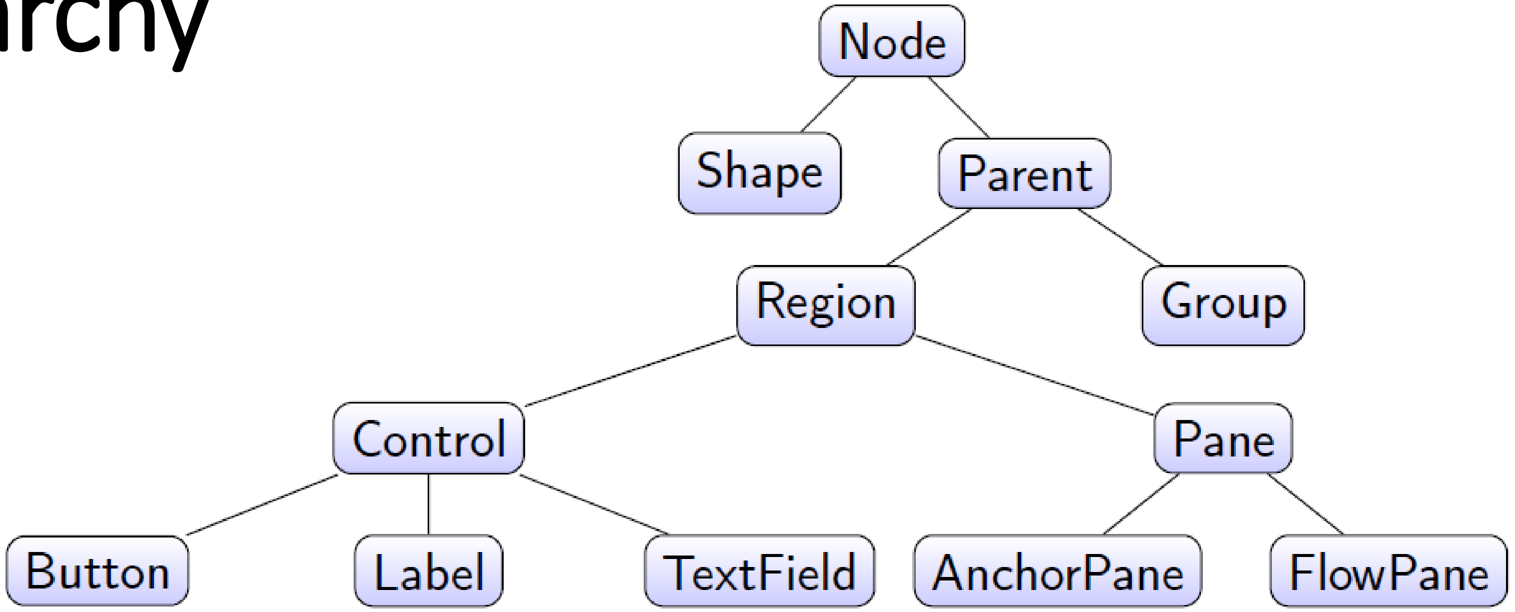
- JavaFX application is divided hierarchically into 3 main components
  - Stage
  - Scene
  - Nodes
- Every JavaFX application is a subclass of javafx.application.Application class which provides the following methods
  - `public void init()`
  - `public abstract void start(Stage primaryStage)`
  - `public void stop()`

JavaFX applications must implement the `start` method

JavaFX Objects

- Scene holds all the physical contents (nodes)
  - Nodes are "visual" components
  - e.g. panes, shapes, images, buttons, etc.
- At one instance, the scene object can only be added to one stage
- JavaFX objects such as Panes, UI controls, and shapes are all subtypes of Node
  - Pane objects help managing the layout of nodes (location and size)
  - UI Controls: buttons, labels, radio buttons, etc.

JavaFX: Class hierarchy



- Pane
  - All Parent nodes have children (held in an internal list)
  - Exposed via: `ObservableList<Node> getChildren()`
  - Add child nodes by adding it to the list
    - E.g. add a button: `root.getChildren().add(button);`

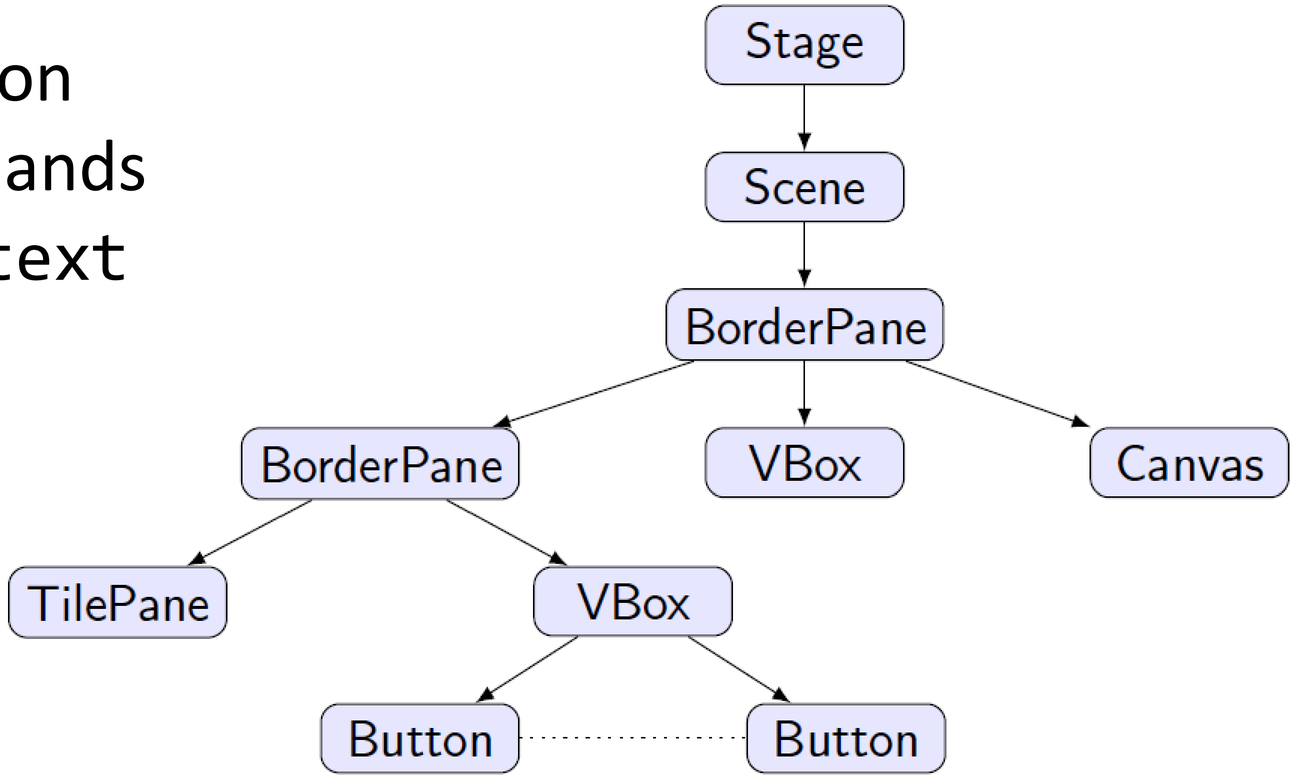
Resizing

- Every node expresses three wishes: minimum, preferred, maximum sizes
  - But layout panes may or may not honour those wishes
- JavaFX offers of a lot of automation wrt (re)-sizing:
  - The size and alignment of nodes is handled by the pane
  - As the pane is resized, the nodes are resized according to their preferred size range preferences
  - By default, UI controls compute default values for their preferred size that is based on the content of the control, but you can set it directly
  - UI controls also provide default minimum and maximum sizes

More on this topic: https://docs.oracle.com/javafx/2/layout/size_align.htm

Scene Graph – Nested layout panes

- Canvas:
  - An image that can be drawn on using a set of graphics commands provided by a `GraphicsContext`



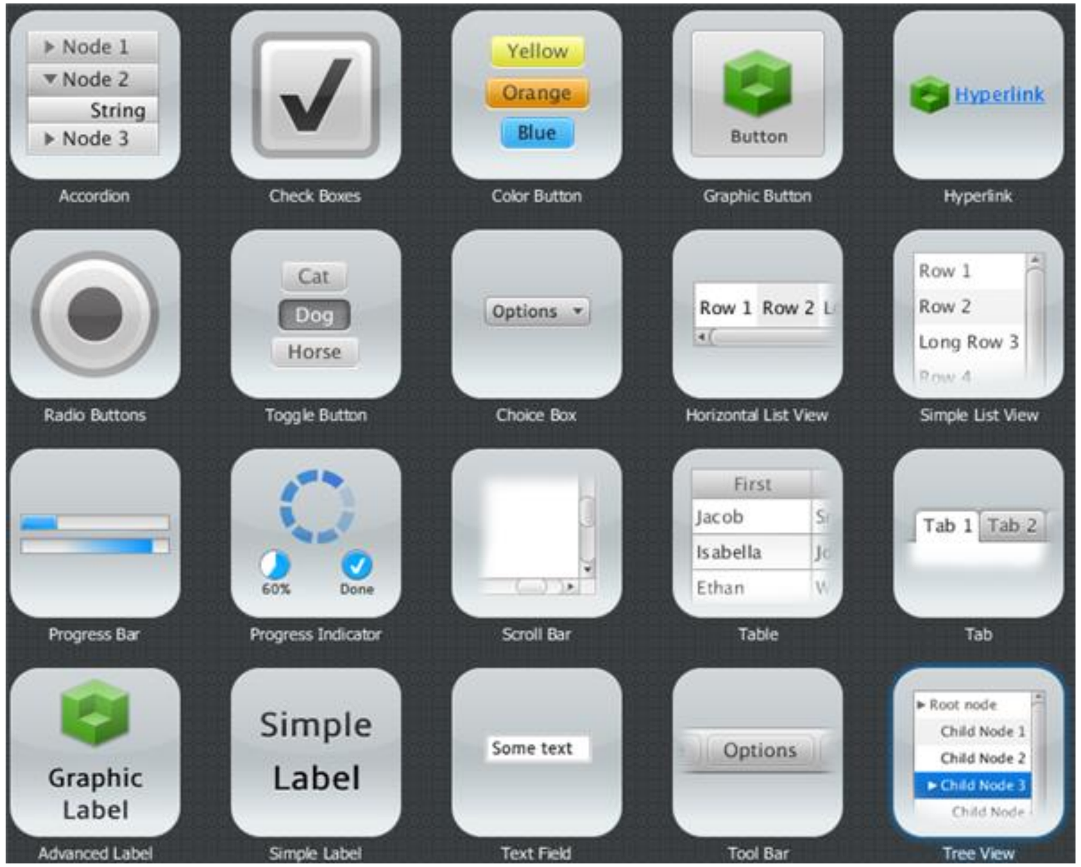
Layout managers

- Some JavaFX panes
  - FlowPane: lays out its children in a flow that wraps at the pane’s boundary
  - GridPane: lays out its children within a flexible grid of rows and columns
  - BorderPane: lays out children in *top*, *left*, *right*, *bottom* and *center* positions
  - TilePane: lays out its children in a grid of *uniformly* sized "tiles”
  - StackPane: lays out its children in a back-to-front stack
  - HBox: lays out its children in a single *horizontal* row
  - VBox: lays out its children in a single *vertical* column

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

UI Controls

- Label
- Button
- Radio Button
- Toggle Button
- Checkbox
- Choice Box
- Text Field
- …



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

JavaFX Dialog

- A Dialog box is a standard window used to interact with the user,
  - e.g., ask the user for confirmation or alert them of an error, etc.
- JavaFX provides the Dialog class and 3 implementations:
  - Alert, e.g., "Are you sure you want to delete the file?”
  - ChoiceDialog to show the user a list of choices and make them pick one
  - TextInputDialog to ask the user for (text) input
- You can also define your own implementation of Dialog

Demo:

Using JavaFX Dialog and FileChooser