Software Design and Analysis
CS-3004
Lecture#10

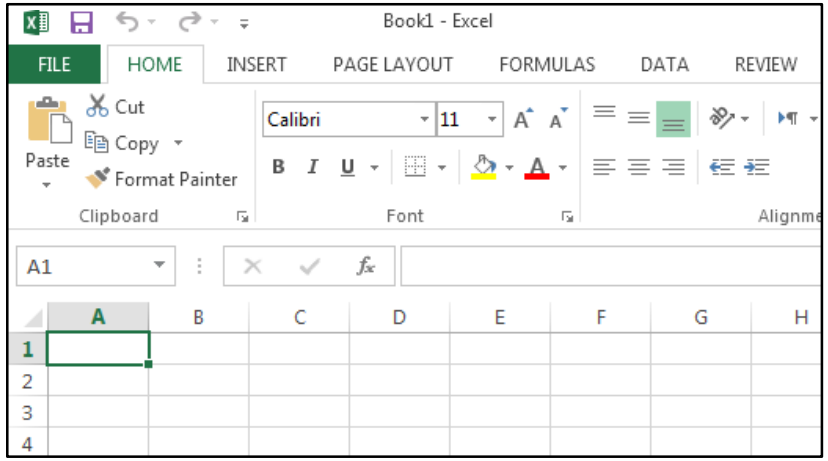
Dr. Javaria Imtiaz,
Mr. Basharat Hussain,
Mr. Majid Hussain

quest lab

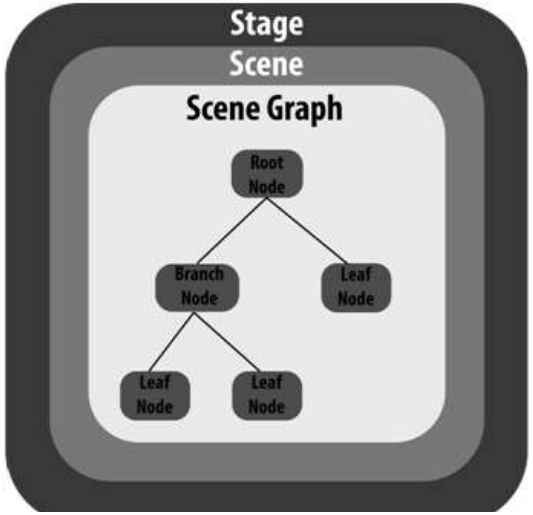
What is GUI in Java?

- GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications.
- It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application.
- GUI plays an important role to build easy interfaces for Java applications.

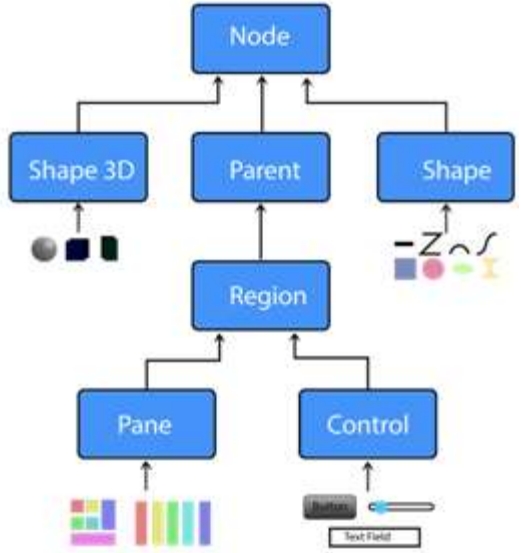
Graphical User Interface



JavaFX



- JavaFX is the latest graphical user interface framework. It is a platform for making a really amazing looking GUI application.
- JavaFX is a Java **library** used to build Rich Internet Applications.
- The applications written using this library can run consistently across multiple platforms.
- The applications developed using JavaFX can run on various devices such as Desktop Computers, Mobile Phones, TVs, Tablets, etc..

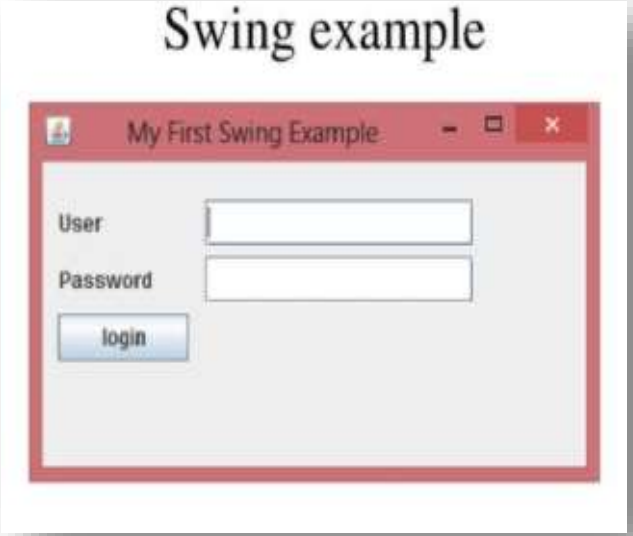


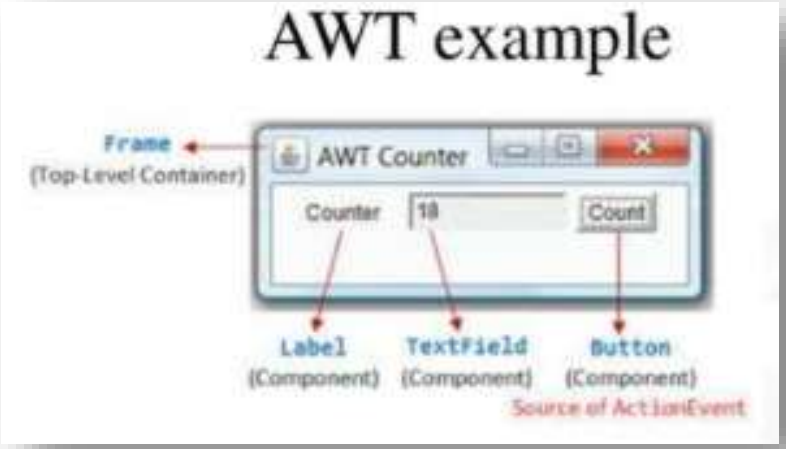
GUI Programming

- To develop **GUI Applications** using Java programming language, the programmers rely on libraries such as **Advanced Windowing Tool kit** and **Swing**.

- After the advent of JavaFX, these Java programmers can now develop GUI applications effectively with rich content.

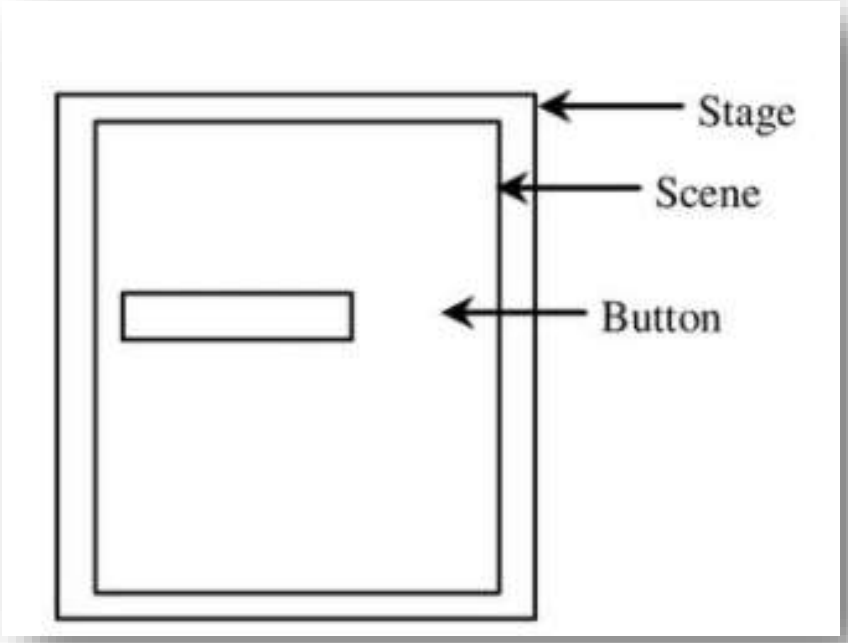
JavaFX vs Swing vs AWT



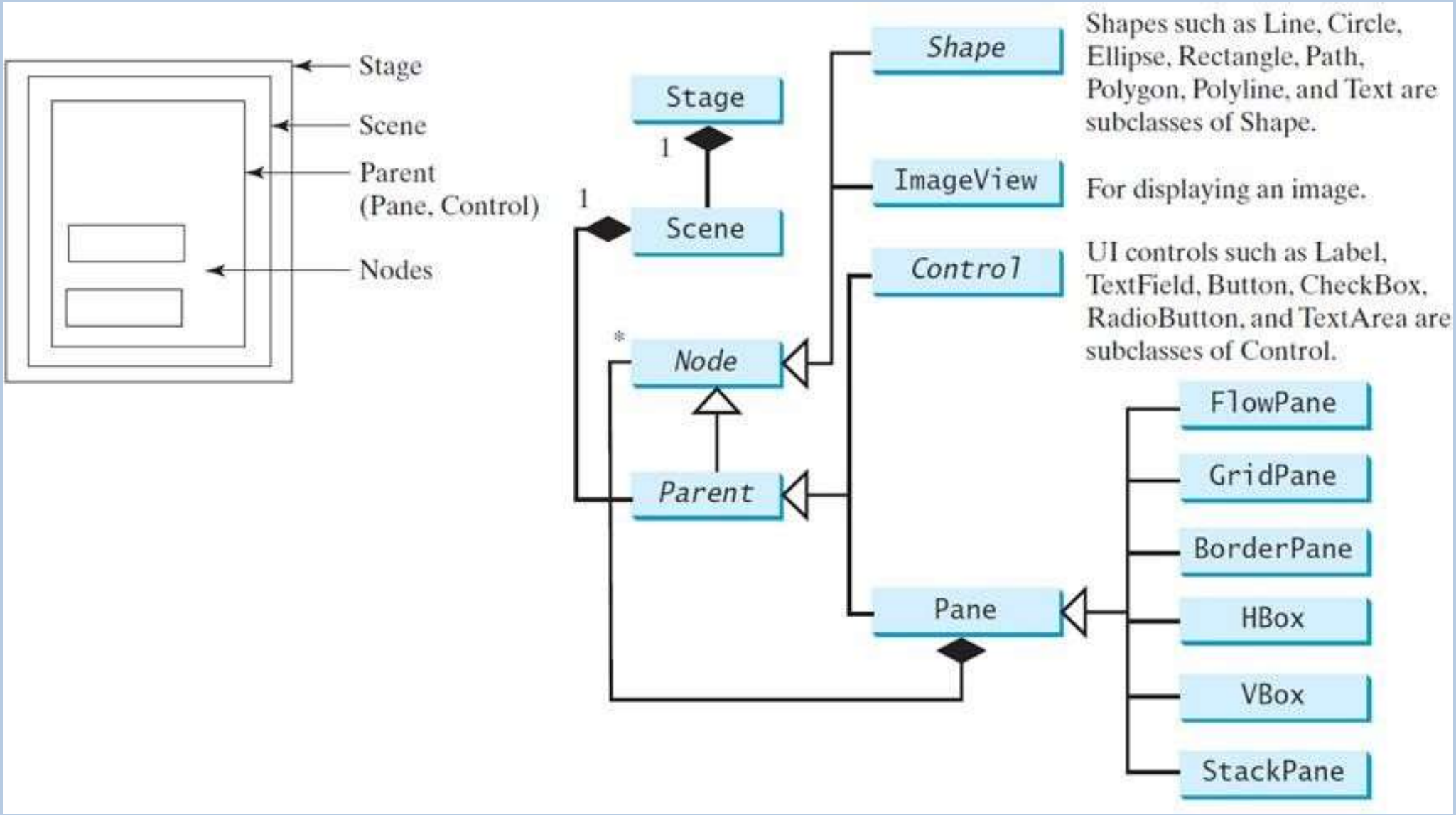


Steps to create a JavaFX Program

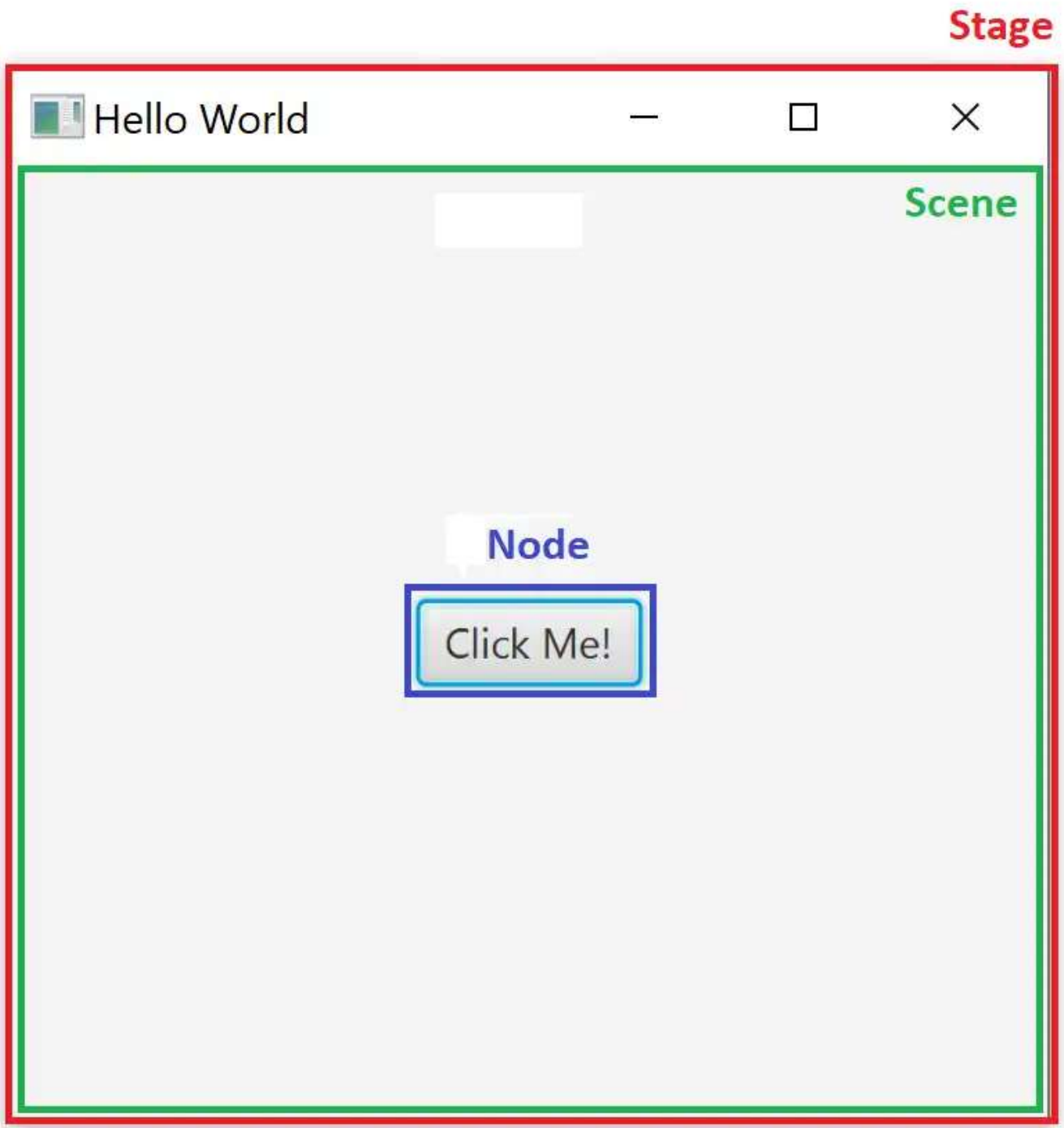
1. Extend **Application**
2. Override **start** (Stage)
3. Create **Nodes** (e.g., Button)
4. Place the Nodes in the **Scene**
5. Place the Scene on **Stage**
6. Show Stage



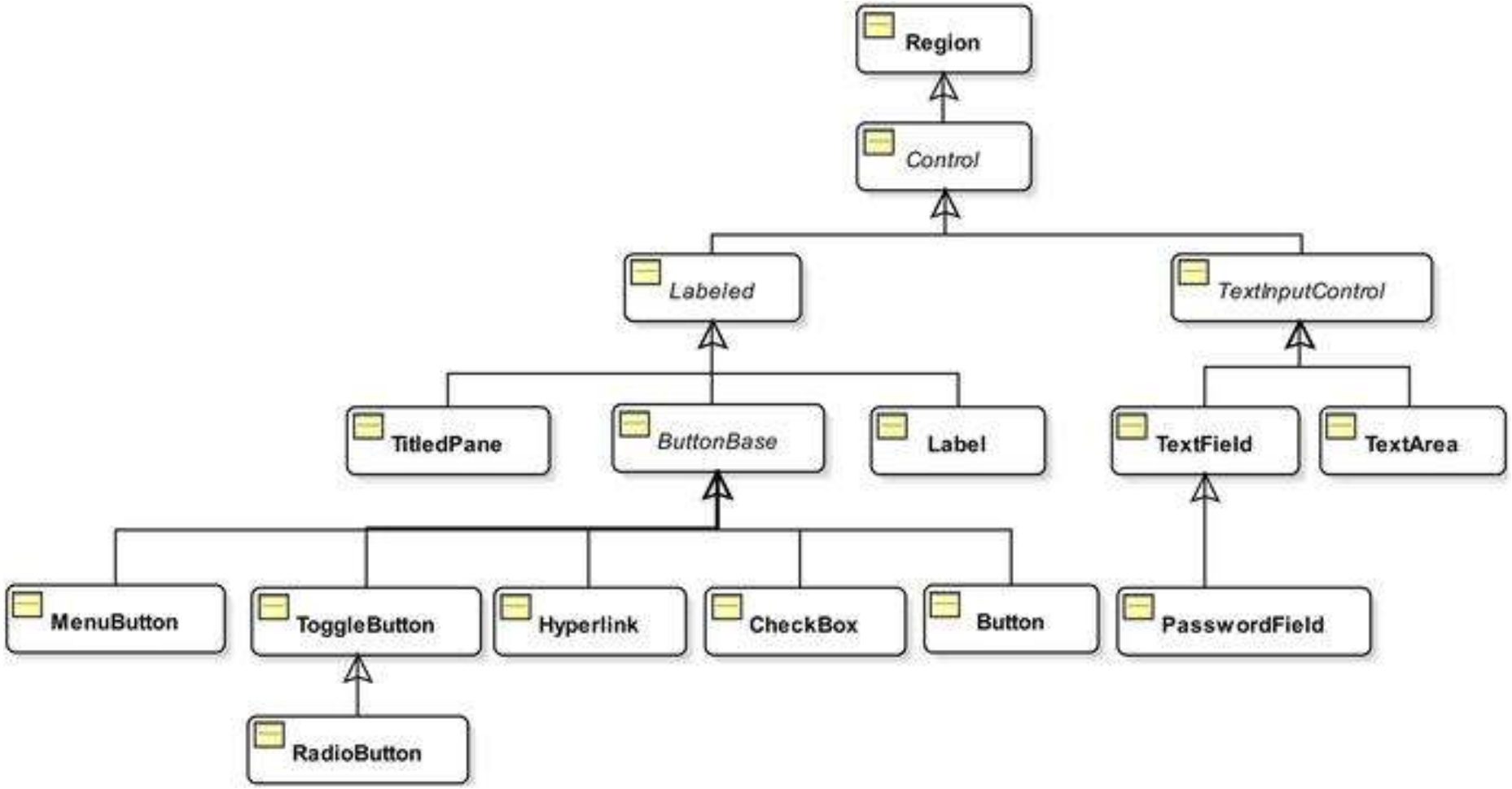
Panes, UI Controls, and Shapes



Example



UI Component



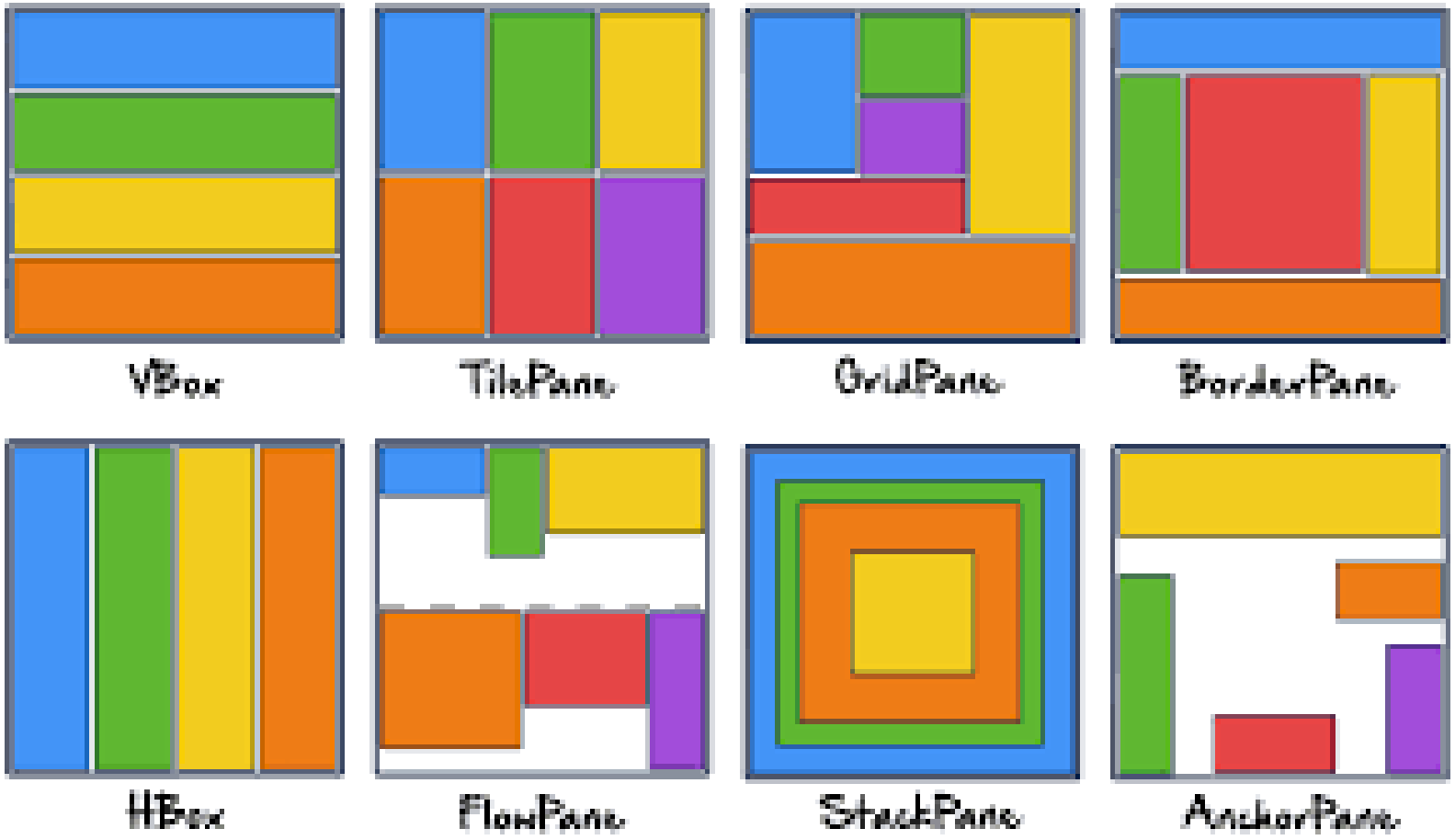
Main Building Blocks

1. Stage
2. Scene
3. Nodes
4. Event Handlers
5. Timelines and transitions for animation purposes

Layout Panes

Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

Example


VBox
TilePane
GridPane
BorderPane
HBox
FlowPane
StackPane
AnchorPane

User Interface Code

- File->new->other->JavaFX project

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;


public class Main extends Application {
    //Inside the main() method, we can launch our application using Application.launch().
    public static void main(String[] args) {
        launch(args);
        }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("My First JavaFX GUI"); //add some nice caption to our window.
        Button btnHello= new Button("Hello"); //Create GUI Elements

        StackPane layout= new StackPane();
        layout.getChildren().add(btnHello);

        Scene scene1= new Scene(layout, 300, 250); //create
        primaryStage.setScene(scene1);

        primaryStage.show(); // It is hidden by default.
        }
        }
```

Main Frame

GUI Widgets

Select Layout

Container

Add scene in a primary stage

Step#1: Download JavaFX SDK

openjfx.io/openjfx-docs/

Apps | Editorial Manager® | Top 40 Data Structu... | New Tab | Time Complexity Ex... | (4516) How to rever... | Introduction to Jav... | Time Table, FAST Sc... | DevOps Tutorial Fo... | Projects - Hom

JavaFX

Getting Started with JavaFX

Introduction

JavaFX allows you to create Java applications with a modern, hardware-accelerated user interface that is highly portable.

There is detailed reference documentation for JavaFX, and this short tutorial will show you how to write a JavaFX 11 application.

For information on how to run JavaFX applications on mobile platforms, please see Getting Started with Gluon Mobile.

For information on Long Term Support (LTS) for JavaFX 11, please see JavaFX Long Term Support options.

JavaFX builds on top of JDK and is a standalone component. There are 2 different options for developing JavaFX applications:

- Use the JavaFX SDK (choosing between 11 LTS, latest release 15 or an early access build).
- Use a build system (e.g. maven/gradle) to download the required modules from Maven Central (choosing as well between the same mentioned versions).

Cont..

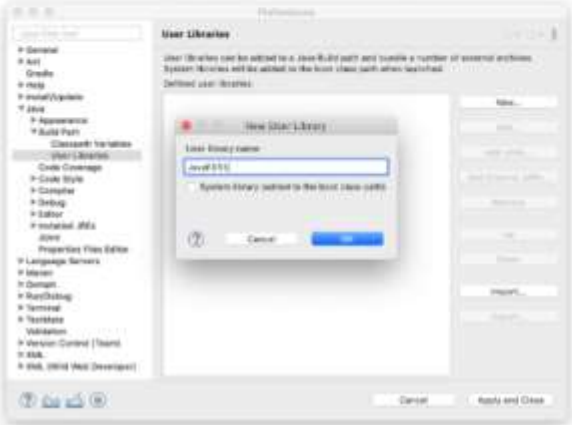
JavaFX

IDE

Follow these steps to create a JavaFX non-modular project and use the IDE tools to build it and run it. Alternatively, you can download a similar project from here.

Download the appropriate JavaFX SDK for your operating system and unzip it to a desired location, for instance `/Users/your-user/Downloads/javafx-sdk-11`.

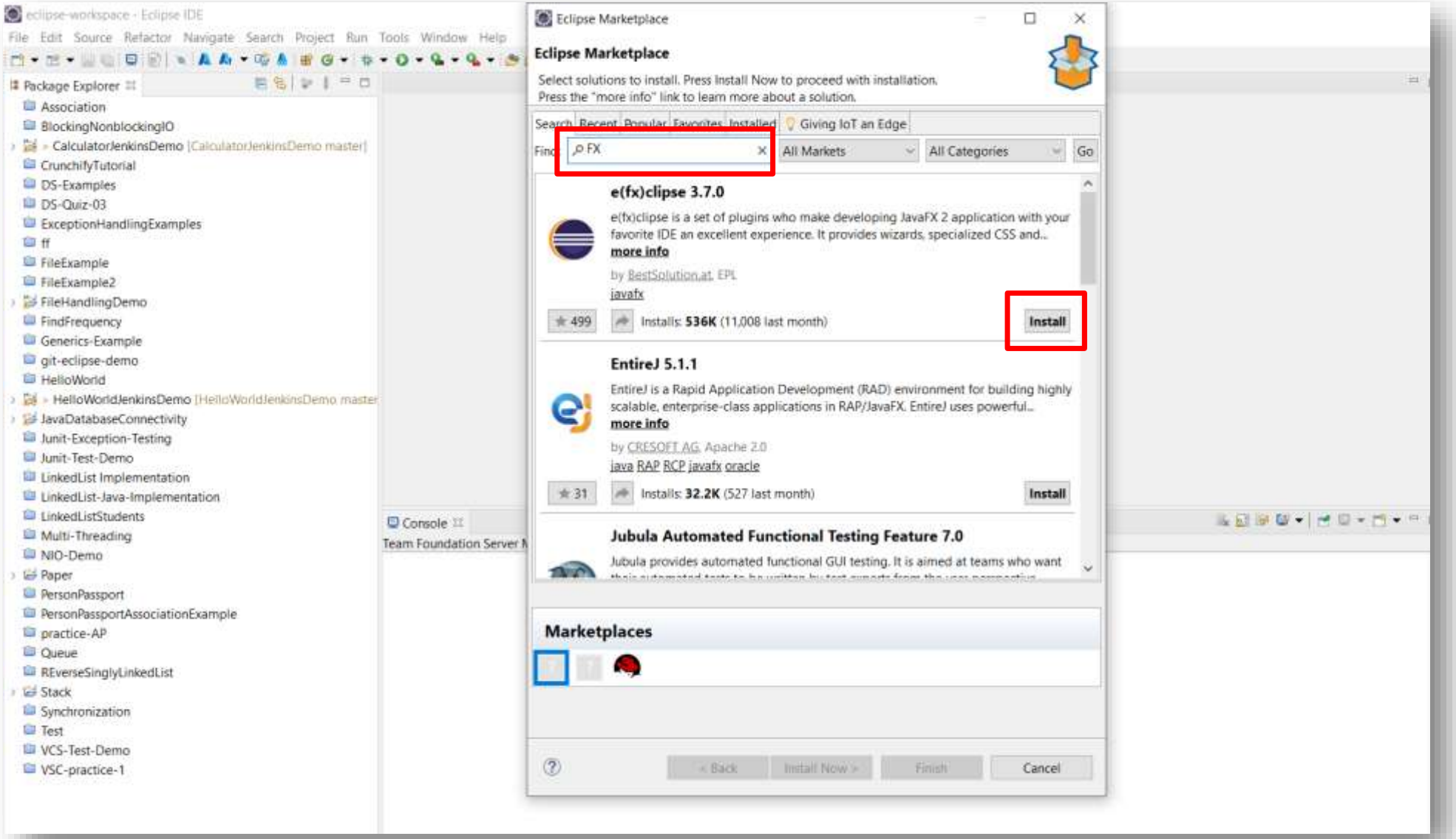
Create a new `User Library` under `Eclipse -> Window -> Preferences -> Java -> Build Path -> User Libraries -> New`.



Cont..

Linux	17.0.1	arm32	SDK	Download [SHA256]
Linux	17.0.1	x64	SDK	Download [SHA256]
Linux	17.0.1	x64	jmods	Download [SHA256]
Linux	17.0.1	x64	Monocle SDK	Download [SHA256]
macOS	17.0.1	aarch64	SDK	Download [SHA256]
macOS	17.0.1	aarch64	jmods	Download [SHA256]
macOS	17.0.1	aarch64	Monocle SDK	Download [SHA256]
macOS	17.0.1	x64	SDK	Download [SHA256]
macOS	17.0.1	x64	jmods	Download [SHA256]
macOS	17.0.1	x64	Monocle SDK	Download [SHA256]
Windows	17.0.1	x64	SDK	Download [SHA256]
Windows	17.0.1	x64	jmods	Download [SHA256]
Windows	17.0.1	x64	Monocle SDK	Download [SHA256]
Windows	17.0.1	x86	SDK	Download [SHA256]

Install JavaFX



Search for e(fx)clipse

eclipse-workspacec - Eclipse IDE

File Edit Source Refactor Navigate Search Project

Package Explorer

- BMS
- demo
- EMF_Demo
- EmployeeFileHandlingDemo
- FileHandlingDemo
- HMS
- Lecture01
- TestProject1

Install

Available Software

Check the items that you wish to install.

Work with: https://download.eclipse.org/releases/2019-03 | Add... | Manage...

e(fx)clipse | Select All | Deselect All

Name	Version
Application Development Frameworks	
e(fx)clipse - Minimal JavaFX OSGi integration bundles	**3.5.0.201902220600**

1 item selected

Details

Minimal feature used to integrate javafx using adapter hooks which is needed when embedding into SWT and/or you are not able to modify the default classloader

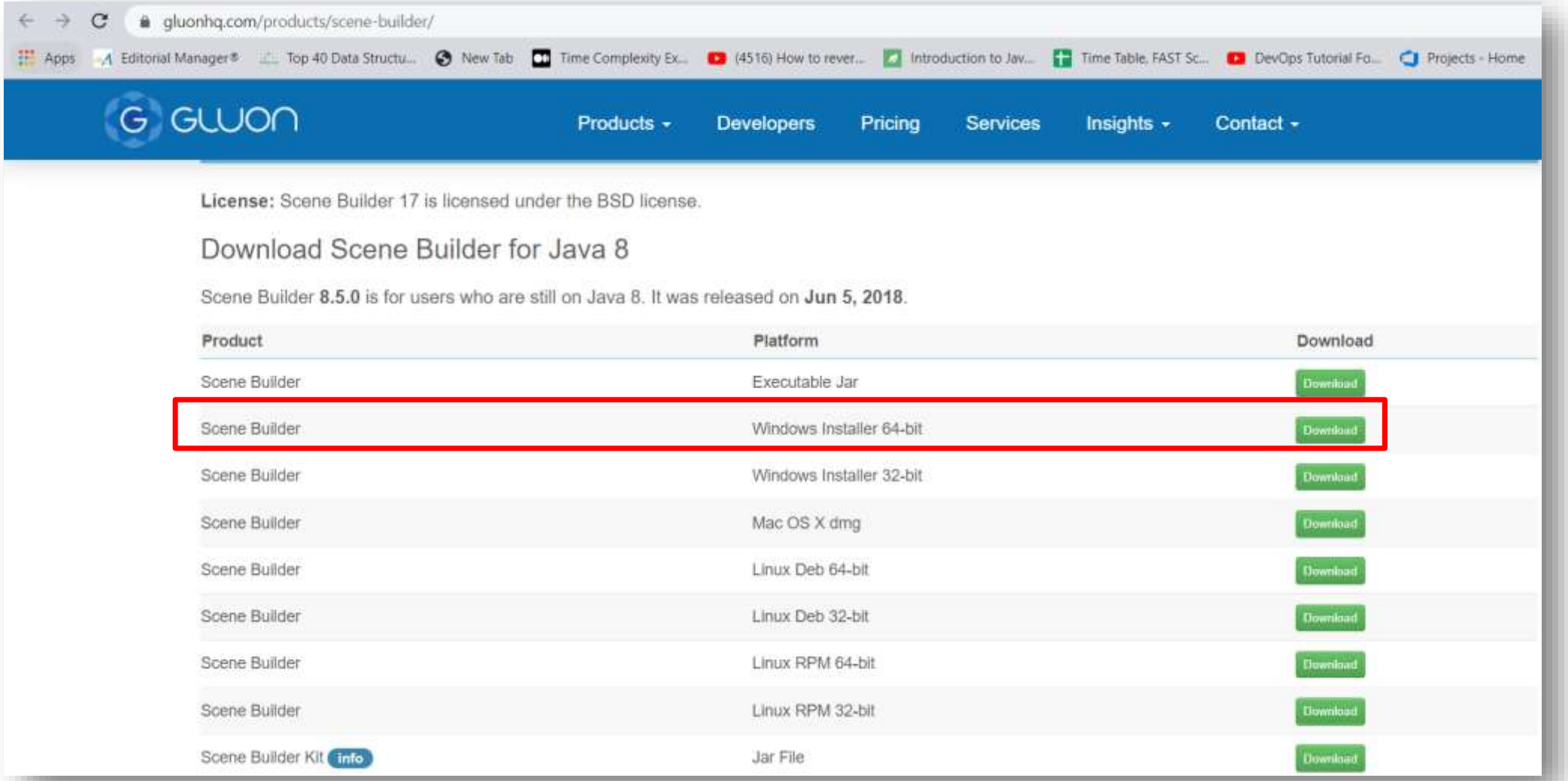
More...

- [x] Show only the latest versions of available software
- [x] Group items by category
- [ ] Show only software applicable to target environment
- [x] Contact all update sites during install to find required software
- [x] Hide items that are already installed

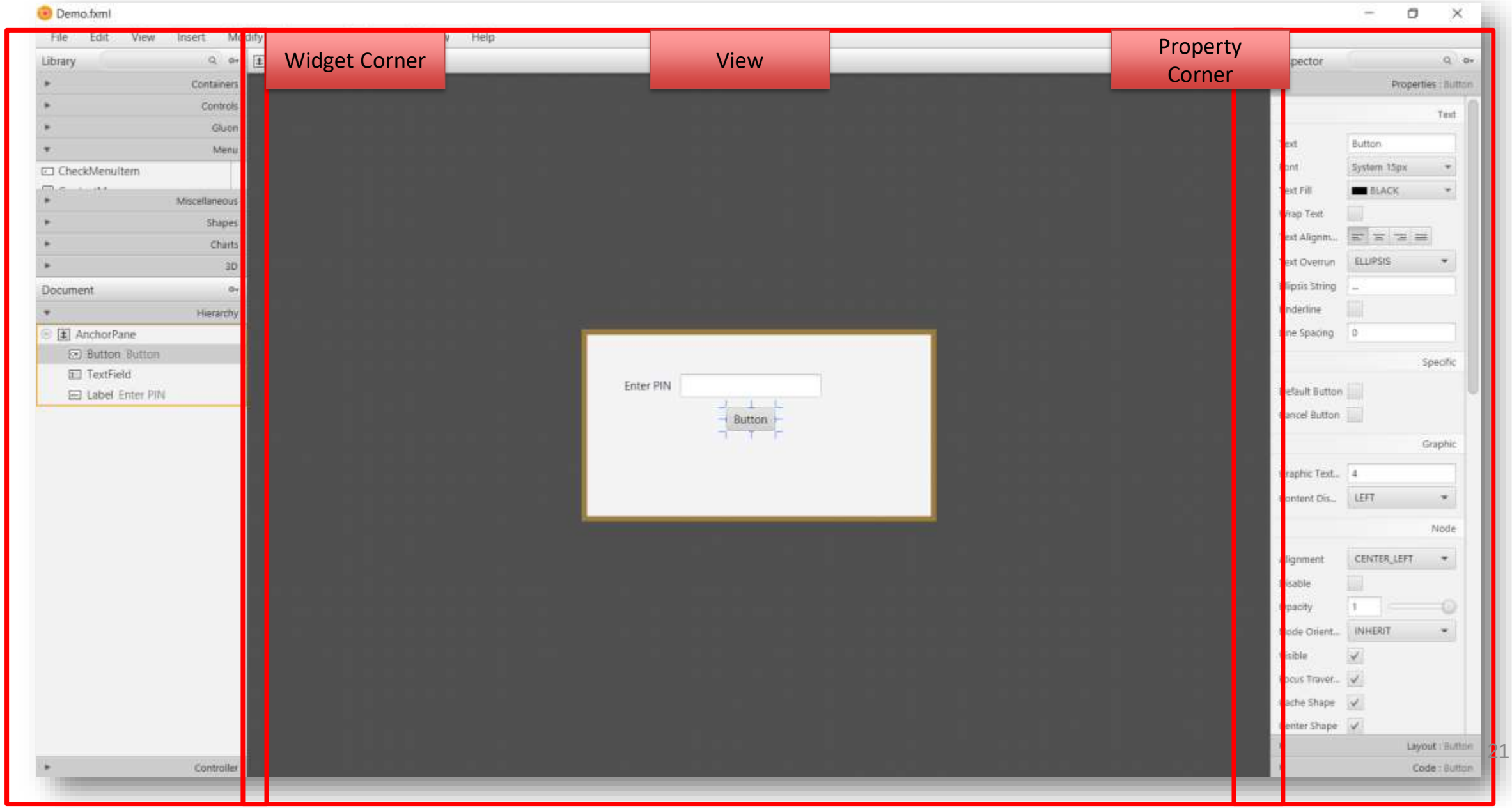
What is already installed?

https://download.eclipse.org/efxclipse/updates-released/3.5.0/site/

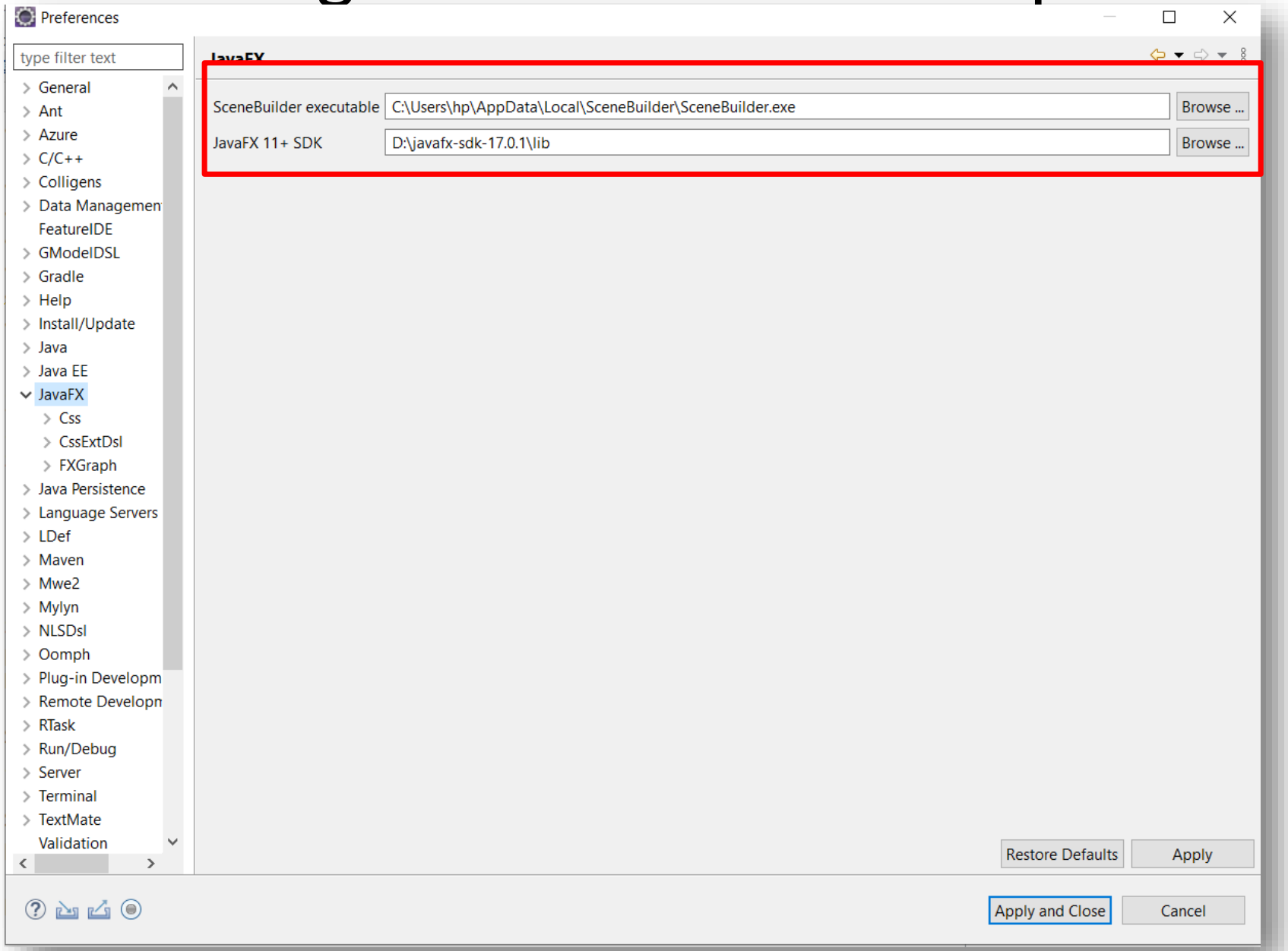
Scene Builder

- 

Scene Builder



Adding Scene Builder in Eclipse

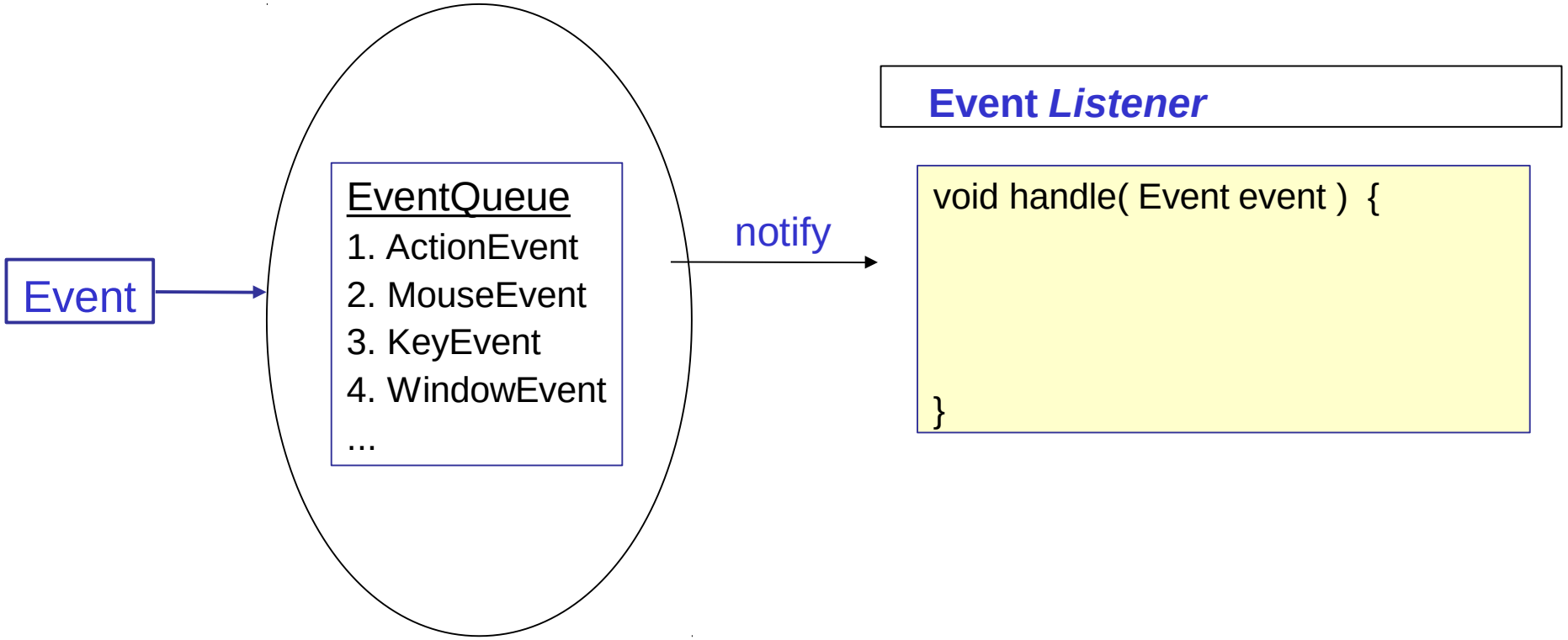


Event Handling in JavaFX

- Writing GUI applications requires that program control be driven by the user's interaction with the GUI.
- When the user moves the mouse, clicks on a button, or selects an item from a menu an “event” occurs.
- The javafx.event package provides the basic framework for FX events.
  - The Event class serves as the base class for JavaFX events.
  - Associated with each event is an event source, an event target, and an event type.

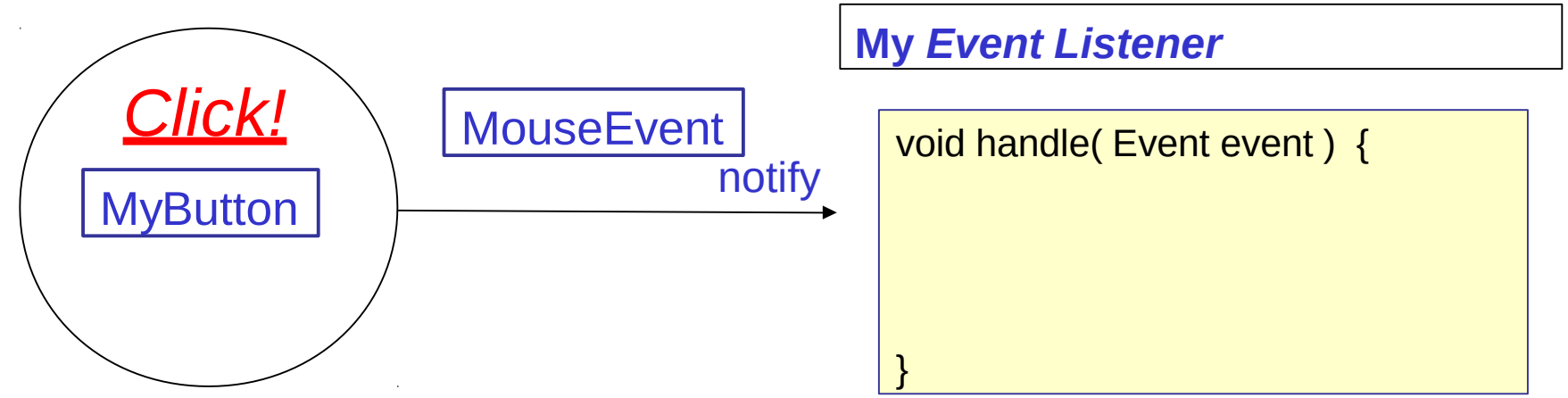
Event Driven Programming

- Graphics applications use ***events***.
- An event dispatcher receives events and ***notifies*** interested objects.



Example

1. User clicks mouse on a button -- that's an ***Event***.
2. JavaFX creates a MouseEvent object.
   - the MouseEvent describes what happened (which mouse button was presses, which field it was in).
3. JavaFX looks for a registered "*Event Listener*", and calls it using the MouseEvent as parameter.



JavaFX user interface

Responding to Behavior

Your application must ***do something*** when an event occurs.

Things you need to know

- what kinds of events are there?
- what user (or software) action causes what event? how do you write an event handler?
- how do you add event handler to a component?

Check the Event class API

All Events are subclasses of Event.

`Event`

- **`ActionEvent`**
- **`InputEvent`**
  - **`KeyEvent`**
  - **`MouseEvent`**
- **`WebEvent`**
- **`WindowEvent`**

Source of Events

- A component or node can be "source" of many kinds of events. Some event types are different for each node or component.
- Its not complicated! Mostly you can *guess* event types.

Button	ActionEvent (button press)
TextField	ActionEvent KeyEvent (Key Press, Key Release, Key Typed).
Any kind of Node	MouseEvent: MousePress, MouseReleased, MouseClicked, MouseDragged, etc. Rotation events, Touch events

What is an EventHandler?

JavaFX has just *one interface* for all kinds of Event Handlers. This is a lot simpler than Swing and AWT.

You have to write code to implement this interface.

<<interface>> **EventHandler<*T extends Event*>**
+handle(event: T): void

Example: ENTER or Button click

1. User types his name and clicks a button (or ENTER)

Event type is: **`ActionEvent`**

```
class ButtonHandler
      implements EventHandler<ActionEvent>
{
   public void handle(ActionEvent evt) {
       String text = nameField.getText();
       // greet user using Alert dialog box
       alert("Hello, "+text);
       nameField.setText("");  // clear input
    }
}
```

How to Add Event Handler

There are two ways.

1) **`addEventHandler`** - the general way

2) **`setOnXxxxx`** - convenience methods for specific event type, such as:

```
setOnAction( EventHandler<ActionEvent> e )
setOnKeyTyped( EventHandler<KeyEvent> e )
setOnMouseClicked( EventHander<MouseEvent> e )
setOnMouseMoved( EventHander<MouseEvent> e )
```

...

2 Ways to Add Event Handler (demo)

```
// 1. use addEventHandler:
        button.addEventHandler(ActionEvent.ALL,new ButtonHandler( ) )

// 2. use setOnAction
        button.setOnAction( new ButtonHandler( ) )
```

- Notice that the EventHandler is the same. The result will be the same, too.
- Both add Event Handler for ActionEvents.

You can re-use event handlers

- For clarity, or to *reuse* the same event handler on many components, assign new event handler to a reference variable first.

- Then use the variable in **`setOnAction(...)`**.

```
ButtonHandler greetHandler = new ButtonHandler();
// Now apply handler to components
button.setOnAction( greetHandler );
nameField.setOnAction( greetHandler );
```

Don't Create Duplicate Handlers

It is bad programming to create two objects to do the same thing (greet the user).

```
// don't do this
button.setOnAction( new ButtonHandler() );
nameField.setOnAction( new ButtonHandler() );
```

4 Ways to Define an EventHandler

1. Define an (inner) class that implements EventHandler. We just did that.

2. Write it as *anonymous class*.

3. Write it as a *method* and use a *method reference*. Method reference is new in Java 8.
   - Works because Event Handler has only 1 method.

4. Write it as a *lambda expression* and use a reference variable to add it.

Event Handler as Anonymous Class

You must specify what interface you are implementing, including type parameter.

```
EventHandler<ActionEvent> buttonHandler =
  new EventHandler<ActionEvent>() {
    // anonymous class definition:
    public void handle(ActionEvent evt) {
      String text = nameField.getText();
      //TODO greet user using Alert box
      nameField.setText("");  // clear input
    }
  };
button.setOnAction( buttonHandler );
```

Avoid inline definition & use

This is hard to understand and hard to maintain.
Avoid it. Define the anonymous class *first*, then use it.

```
// This is harder to understand, especially
// when the anonymous class is long.
button.setOnAction(
  new EventHandler<ActionEvent>() {
    public void handle(ActionEvent evt) {
      String text = nameField.getText();
      //TODO greet user using Alert box
      nameField.setText("");  // clear input
    }
  } );
```

Method as Event Handler?

Using SceneBuilder to assign event handlers
we did not write inner classes or anonymous classes.

We just wrote a method, like this:

```
@FXML
public void greetTheUser(ActionEvent evt) {
    String text = nameField.getText();
    //TODO greet user using Alert box
    nameField.setText("");  // clear input
}
```

SceneBuilder let us use a method as Event Handler, instead of object.

How?

Method Reference as EventHandler

Write a method with the required method signature, but any name you like.

```
// Assign event handler using method reference
  button.setOnAction( this::greetAction );

  // this method signature "looks like" an
  // EventHandler, but the name is different
  public void greetAction(ActionEvent evt) {
    String text = nameField.getText();
    //TODO greet user using Alert box
    nameField.setText("");  // clear input
 }
```

Lambda Expressions

Lambda Expression is an inline method definition, without a method name.

```
EventHandler<ActionEvent> buttonHandler =
    (event) -> {
            String text = nameField.getText();
            //TODO greet user using Alert box
            nameField.setText("");
        } ;
button.setOnAction( buttonHandler );
```

5th Way to Define Event Handler

You can define the controller itself as "implements EventHandler<T>" and use "setOnAction(this)".

```
class GreetController
      implements EventHandler<ActionEvent> {
    @FXML
    public void initialize() {
       button.setOnAction( this );
    ...

    public void handle(ActionEvent event) {
       // handle it.
```

This technique is not usually the best choice. You usually have many components which need custom event handlers.

Code for **`alert()`**

```
/**
 * Display a dialog box with a string message.
 * @param message the message to show.
 */
public void alert(String message) {
    Alert alert = new Alert(AlertType.INFORMATION);
    alert.setContentText( message );
    alert.show( );
}
```