Graphic User Interface (GUI) & Basic Listener

Outlines

- History
- JavaFX components
- Starting using GUI
- Basic structure (stage, scene, scene graph, node)
- Layout
- Chart
- Scene builder
- FXML
- Style
- Binding properties
- Basic event handling

History

- AWT is Java's original set of classes for building GUIs
  - Abstract Window Toolkit (AWT)
  - import java.awt.*
  - Uses peer components of the OS; heavyweight
  - Not truly portable: looks different and lays out inconsistently on different OSs
    - Due to OS's underlying display management system
- Swing is designed to solve AWT's problems
  - import javax.swing.*
  - Extends AWT
  - 99% java; lightweight components
  - Layout consistently on all OSs
  - Uses AWT event handling

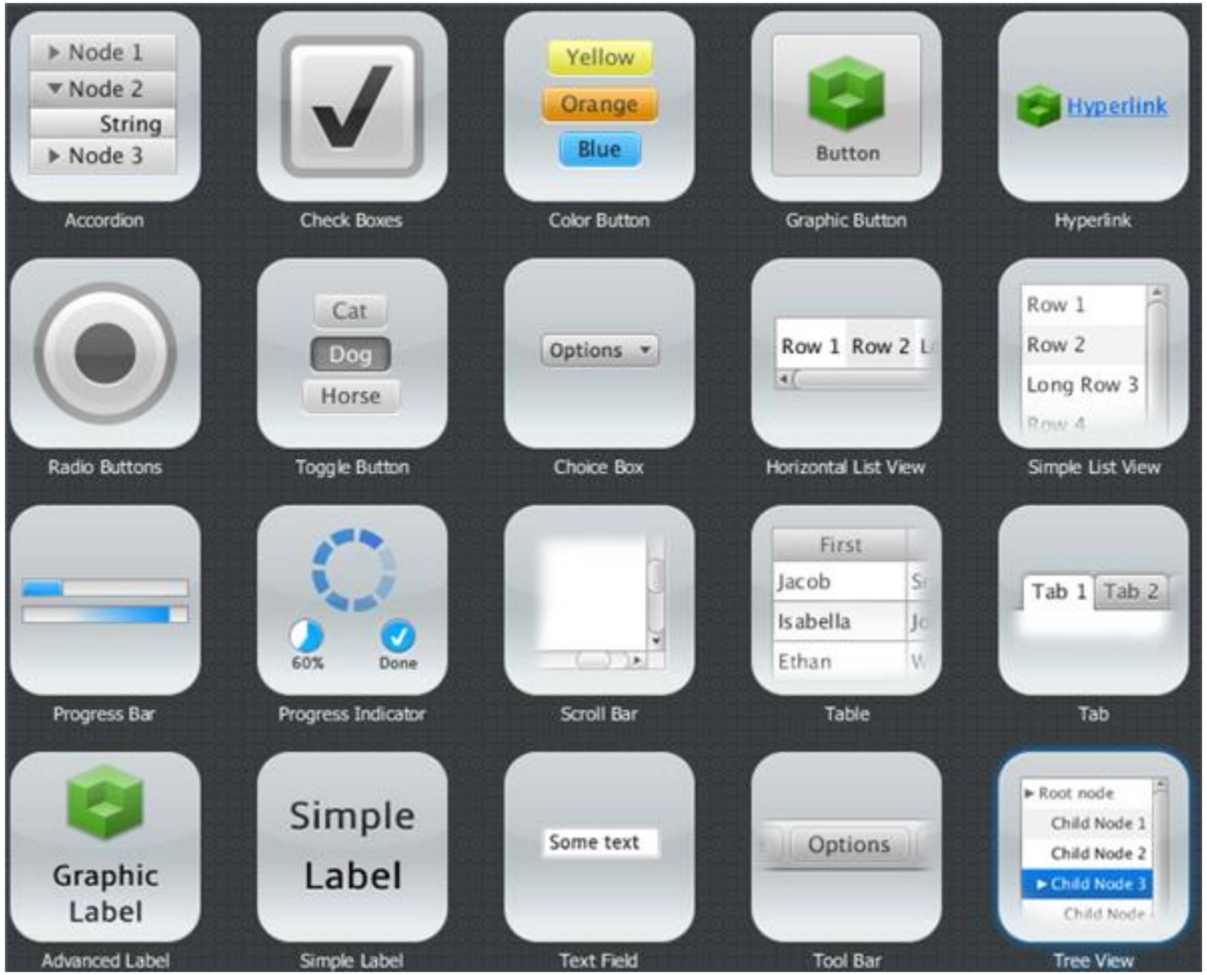
History (cont.)

- JavaFX
  - JAVA + FLASH + FLEX
  - An API included in Java SE 8 for UI development
  - The successor of Java Swing
  - 100% java; lightweight component
  - Swing Node (embed Swing in JavaFX)
  - More features
    - Data binding
    - FXML (mark-up language for designing UI)
    - CSS
    - Charts.
    - 3D Support
    - Etc.
- We will learn JavaFX in this class

JavaFX components

- Containers
  - Anchor Pane, Stack Pane, Tab Pane, HBox, Vbox, ...
- UI Controls
  - Accordion, Label, Button, RadioButton, CheckBox, TextField, TextArea, Slider, Tooltip, ComboBox, ProgressBar, DatePicker, ColorPicker, ...
- Shapes
  - Line, Rectangle Ellipse, Path, Circle Arc, Polygon Polyline, Curve, Text
- Charts
  - LineChart, PieChart, AreaChart, BarChart, ScatterChart, BubbleChart

JavaFX components (cont.)

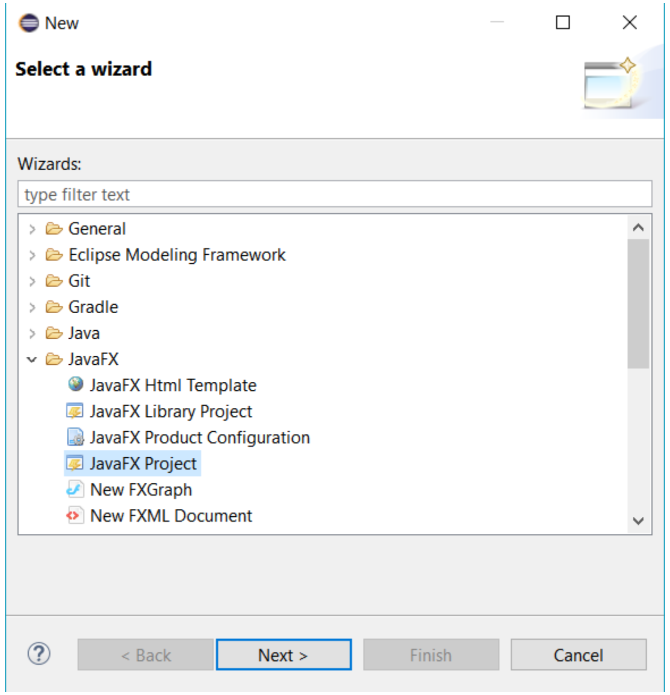


Reference: http://docs.oracle.com/javafx/2/ui_controls/overview.htm

Install E(fx)clipse into Eclipse
http://o7planning.org/en/10619/install-efxclipse-into-eclipse

Create JavaFX project in Eclipse

- Go to File > New > Other
- Select JavaFX > JavaFXProject
- Fill Project Name
- Finish



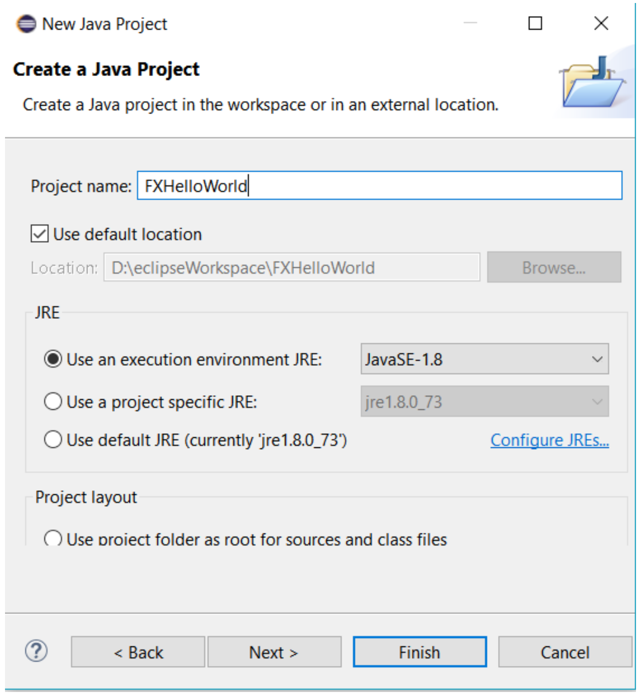
New

Select a wizard

Wizards:

type filter text

- General
- Eclipse Modeling Framework
- Git
- Gradle
- Java
- JavaFX
  - JavaFX Html Template
  - JavaFX Library Project
  - JavaFX Product Configuration
  - JavaFX Project
  - New FXGraph
  - New FXML Document



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: FXHelloWorld

☑ Use default location

Location: D:\eclipseWorkspace\FXHelloWorld

JRE

● Use an execution environment JRE: JavaSE-1.8

○ Use a project specific JRE: jre1.8.0_73

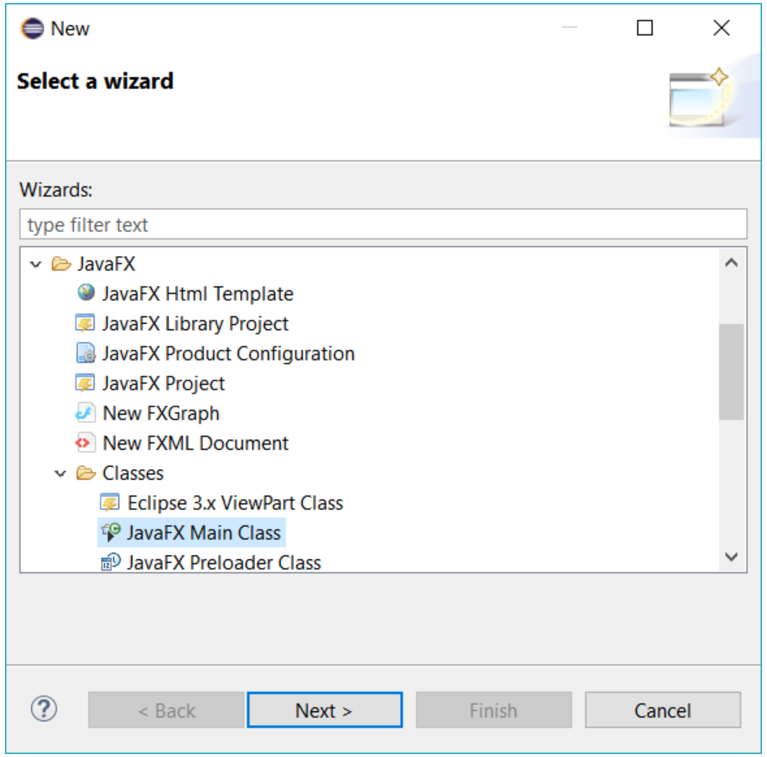
○ Use default JRE (currently 'jre1.8.0_73')

Project layout

○ Use project folder as root for sources and class files

Create JavaFX Class

- New > other
- javaFX > Classes > JavaFX Main Class



New

Select a wizard

Wizards:

type filter text

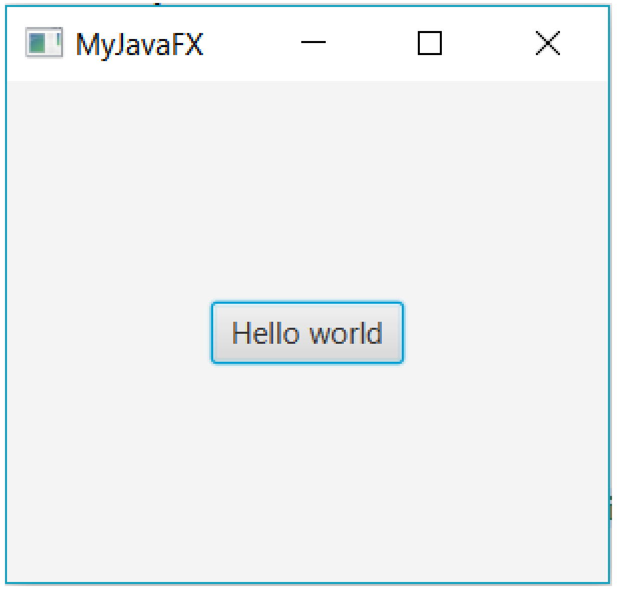
- JavaFX
  - JavaFX Html Template
  - JavaFX Library Project
  - JavaFX Product Configuration
  - JavaFX Project
  - New FXGraph
  - New FXML Document
  - Classes
    - Eclipse 3.x ViewPart Class
    - JavaFX Main Class
    - JavaFX Preloader Class

JavaFX HelloWorld Example

FXHelloWorld.java

```
package application;

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

public class FXHelloWorld extends Application {
      // Override the start method in the Application class
      @Override
      public void start(Stage primaryStage) {
            // Create a scene and place a button in the scene
            Button btn = new Button("Hello world");
            StackPane root = new StackPane();
            root.getChildren().add(btn);
            Scene scene = new Scene(root, 300, 250);
            primaryStage.setTitle("MyJavaFX"); // Set the stage title
            primaryStage.setScene(scene); // Place the scene
            primaryStage.show();
      }

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

May not compile at all!



Exception : The type 'Button' is not API

- Go into the project's build path and edited the JRE System Library, the Java 8 execution environment was selected.
- Choose to use an "Alernate JRE“ then it will fix this error for you.

Accessibility problem
in Eclipse Oxygen

Exception : The type 'Button' is not API

solution 2 is to change the access restrictions.

- Go to the properties of your Java project,
  - i.e. by selecting "Properties" from the context menu of the project in the "Package Explorer".
- Go to "Java Build Path", tab "Libraries".
- Expand the library entry
- select
  - "Access rules",
  - "Edit..." and
  - "Add..." a "Resolution: Accessible" with a corresponding rule pattern. For example:

    javafx/**

JavaFX HelloWorld Example (cont.)

```java
package application;

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

public class FXHelloWorld extends Application {
    // Override the start method in the Application class
    @Override
    public void start(Stage primaryStage) {
        // Create a scene and place a button in the scene
        Button btn = new Button("Hello world");
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene
        primaryStage.show();
    }

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

To create JavaFX application,

- Extends Application (javafx.application.Application)

JavaFX HelloWorld example (cont.)

```java
package application;

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

public class FXHelloWorld extends Application {
	// Override the start method in the Application class
	@Override
	public void start(Stage primaryStage) {
		// Create a scene and place a button in the scene
		Button btn = new Button("Hello world");
		StackPane root = new StackPane();
		root.getChildren().add(btn);
		Scene scene = new Scene(root, 300, 250);
		primaryStage.setTitle("MyJavaFX"); // Set the stage title
		primaryStage.setScene(scene); // Place the scene
		primaryStage.show();
	}

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

To create JavaFX application,

- Extends Application (javafx.application.Application)
- Override the start() method

JavaFX HelloWorld example (cont.)

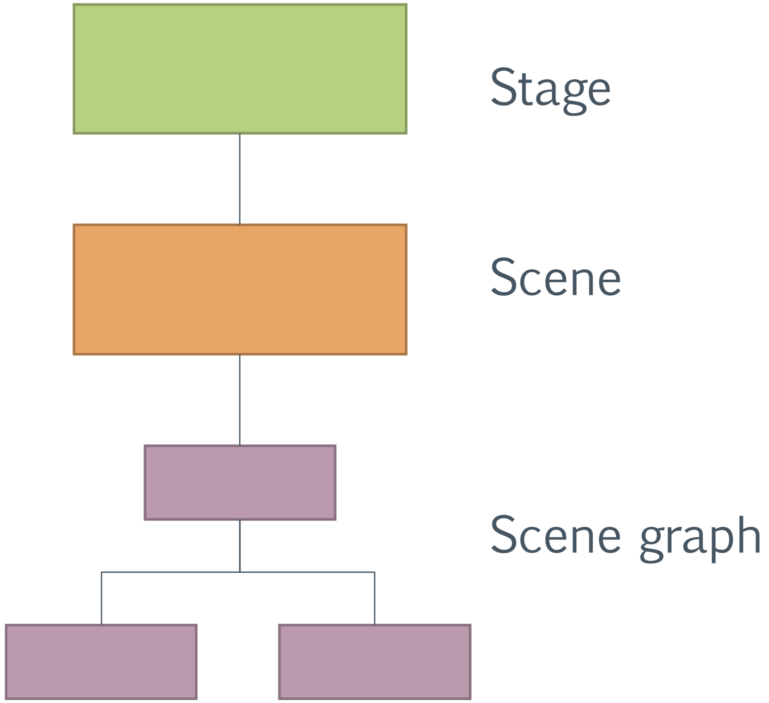
```
package application;

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

public class FXHelloWorld extends Application {
    // Override the start method in the Application class
    @Override
    public void start(Stage primaryStage) {
        // Create a scene and place a button in the scene
        Button btn = new Button("Hello world");
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene
        primaryStage.show();
    }

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

To create JavaFX application,

- Extends Application (javafx.application.Application)
- Override the start() method
- Call launch() (Application.launch())
  - The framework internals call the start() method to start
  - Then, javafx.stage.Stage object is available to use

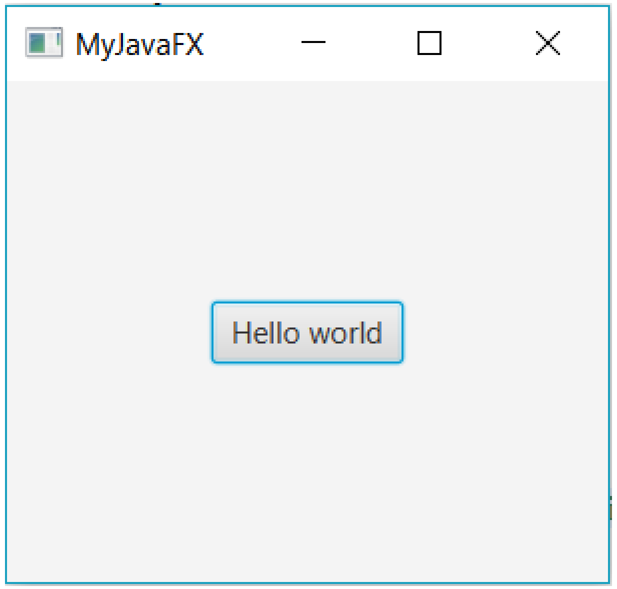
Basic structure

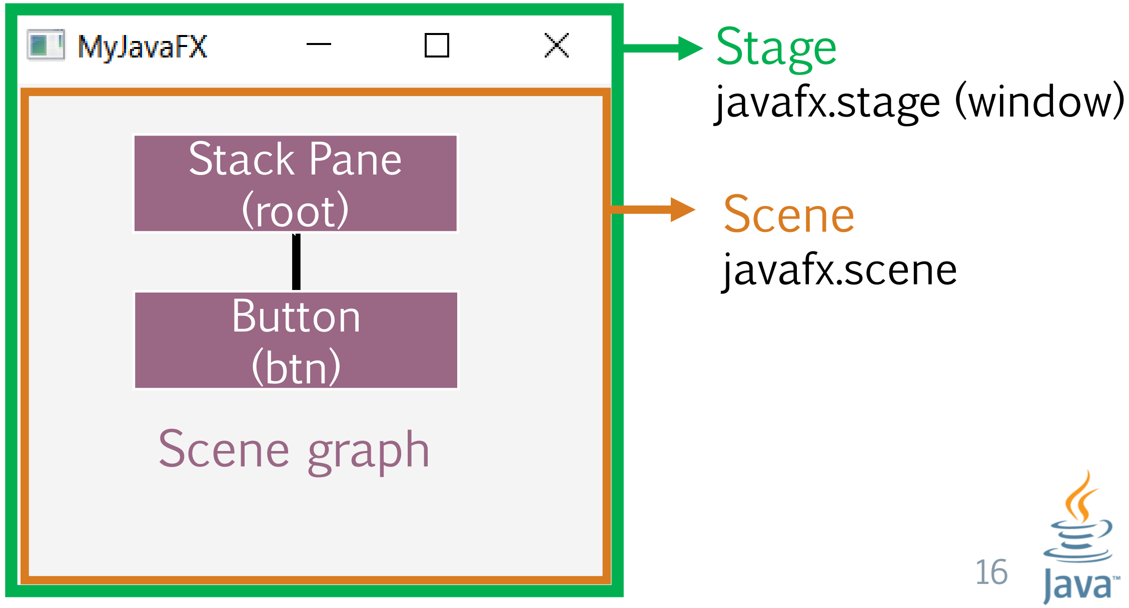
- JavaFX application contains one or more stages which corresponds to windows
- Each stage has a scene
- Each scene can has scene graph (hierarchical tree of nodes)
- Node (UI Components such as control, layout)



JavaFX HelloWorld Example

```
package application;

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

public class FXHelloWorld extends Application {
     // Override the start method in the Application class
     @Override
     public void start(Stage primaryStage) {
          // Create a scene and place a button in the scene
          Button btn = new Button("Hello world");
          StackPane root = new StackPane();
          root.getChildren().add(btn);
          Scene scene = new Scene(root, 300, 250);
          primaryStage.setTitle("MyJavaFX"); // Set the stage title
          primaryStage.setScene(scene); // Place the scene
          primaryStage.show();
     }

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



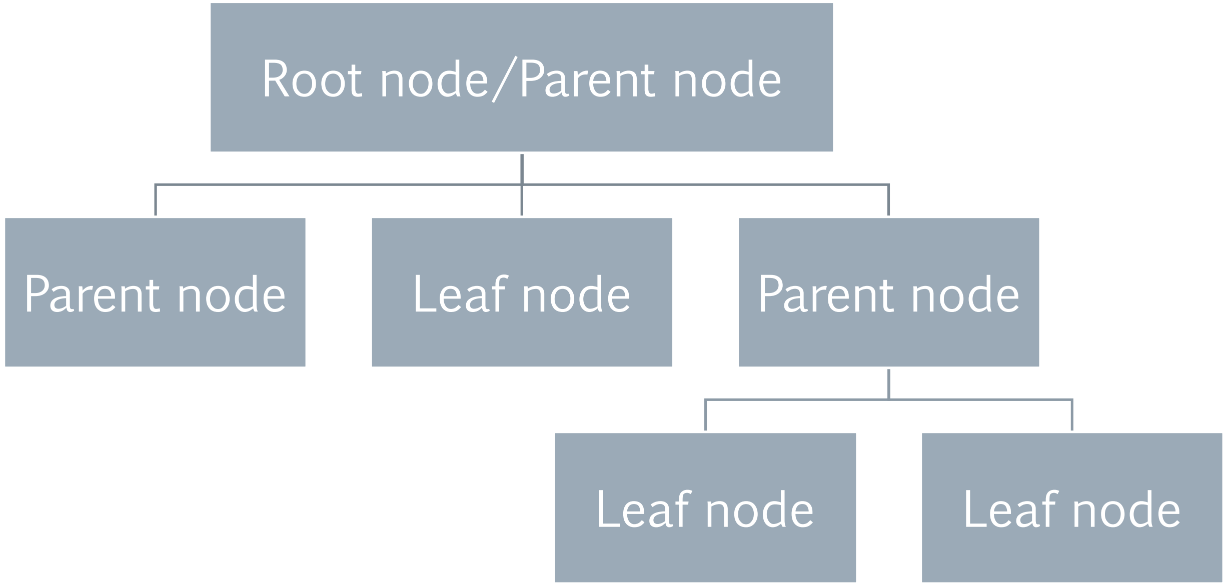


Stage
javafx.stage (window)

Scene
javafx.scene

Scene graphs

- In JavaFX, contents (such as text, images, and UI controls) are organized using a tree-like data structure known as scene graph
- A scene graph is hierarchical tree of nodes



Nodes

- GUI component object, such as geometric shapes, UI controls, layout panes, and 3D objects.
- 3 types of nodes
  - Root Node
    - Parent of all other nodes
    - Scene graph can have only one root node.
  - Parent Node (group of nodes)
    - Can have other nodes as children
  - Leaf Node
    - Cannot have children
    - Not container

Nodes (cont.)

- Node can have the following:
  - ID
  - Style
  - Class
  - Bounding volume
  - Effects such as blurs and shadows
  - Event handlers (such as mouse, keyboard)
- Add nodes to parent

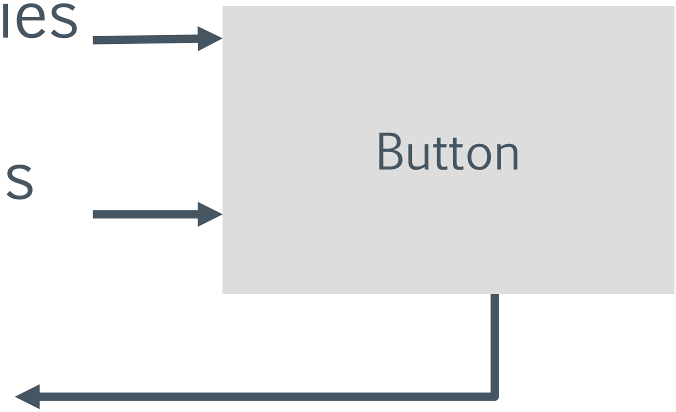
```
myParent.getChildren().add(childNode);
```

or

```
myParent.getChildren().addAll(childNode1, childNode2);
```

Using GUI Component

- Java: GUI component = class
- Properties
- Methods
- Events



Using a GUI component

- 1. Create it

  Button btn = new Button("Hello world");

- 2. Configure it

  // using getter/setter to access properties (text)

  btn.setText(“Hello world”); // methods

- 3. Add it to parent

  root.getChildren().add(btn);

- 4. Listen to it

  Events: Listeners

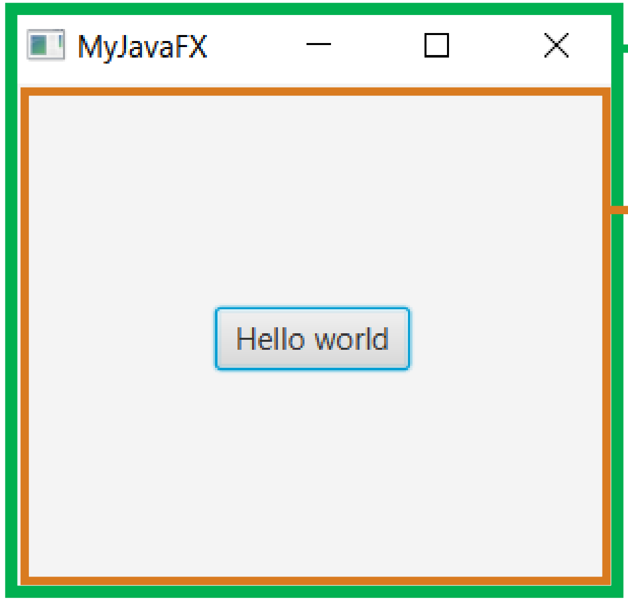
Using a GUI Component

1. Create it
2. Configure it
3. Add children (if root or parent node (container))
4. Add to parent (if not root node)
5. Listen to it

order important

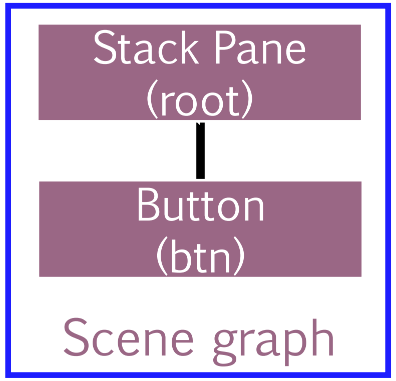
JavaFX HelloWorld Example

```java
package application;

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

public class FXHelloWorld extends Application {
     // Override the start method in the Application class
     @Override
     public void start(Stage primaryStage) {
          // Create a scene and place a button in the scene
          Button btn = new Button("Hello world");
          StackPane root = new StackPane();
          root.getChildren().add(btn);
          Scene scene = new Scene(root, 300, 250);
          primaryStage.setTitle("MyJavaFX"); // Set the stage title
          primaryStage.setScene(scene); // Place the scene
          primaryStage.show();
     }

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



Stage
javafx.stage (window)

Scene
javafx.scene



Scene

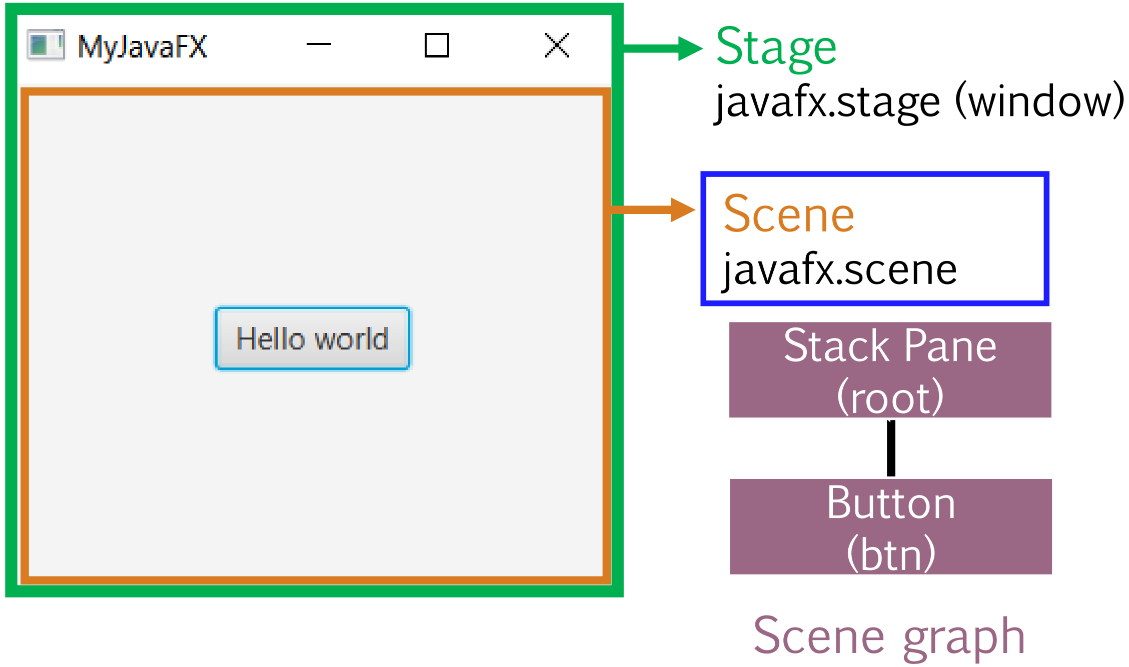
- Container for all contents in a scene graph
- Root node of the scene graph is required for creating Scene

```
Scene scene = new Scene(root, 300, 250);
```

- Be able to set size, color etc.
- If size is not specified, automatically compute based on its contents

JavaFX HelloWorld Example

```
package application;

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

public class FXHelloWorld extends Application {
     // Override the start method in the Application class
     @Override
     public void start(Stage primaryStage) {
          // Create a scene and place a button in the scene
          Button btn = new Button("Hello world");
          StackPane root = new StackPane();
          root.getChildren().add(btn);
          Scene scene = new Scene(root, 300, 250);
          primaryStage.setTitle("MyJavaFX"); // Set the stage title
          primaryStage.setScene(scene); // Place the scene
          primaryStage.show();
     }

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



Stage

- javafx.stage package
- Top level container of the application.
- Usually, OS Window.
- The main stage is created as part of the application launch and passed as an argument in start method

```
public void start(Stage primaryStage)
```

- Be able to set title, size, icon etc.
- Single application can have multiple stages

Stage (cont.)

- Set Stage title

```
primaryStage.setTitle("MyJavaFX");
```

- Set scene to stage

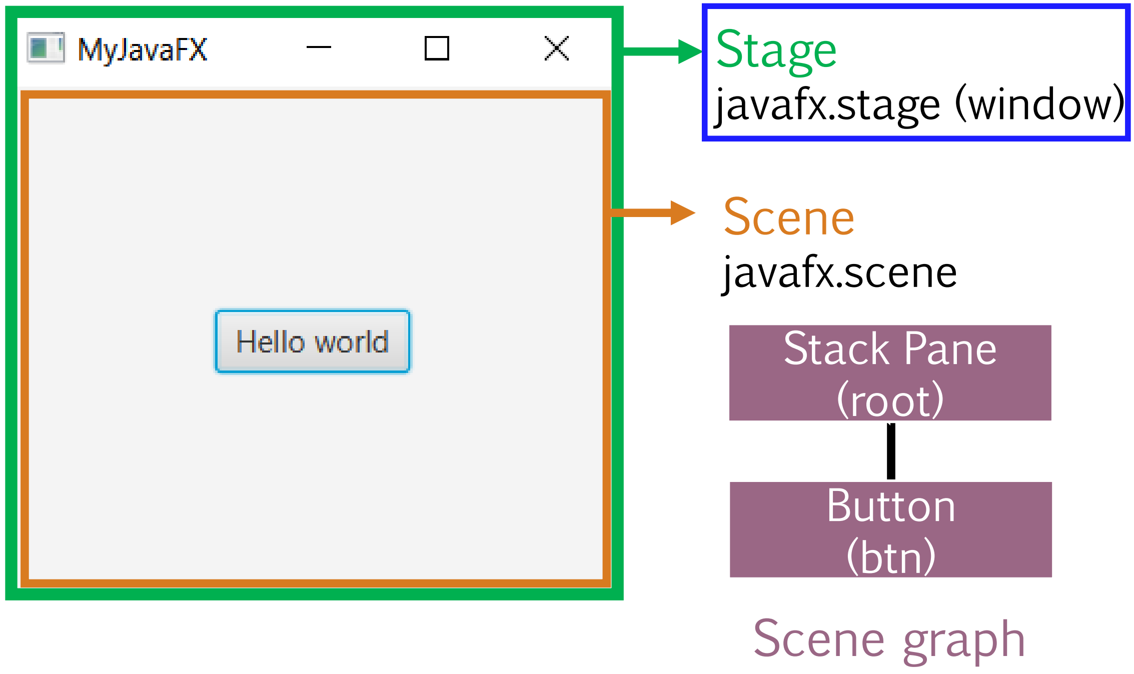
```
primaryStage.setScene(scene);
```

- Show the stage

```
primaryStage.show();
```

JavaFX HelloWorld Example

```java
package application;

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

public class FXHelloWorld extends Application {
    // Override the start method in the Application class
    @Override
    public void start(Stage primaryStage) {
        // Create a scene and place a button in the scene
        Button btn = new Button("Hello world");
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene
        primaryStage.show();
    }

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



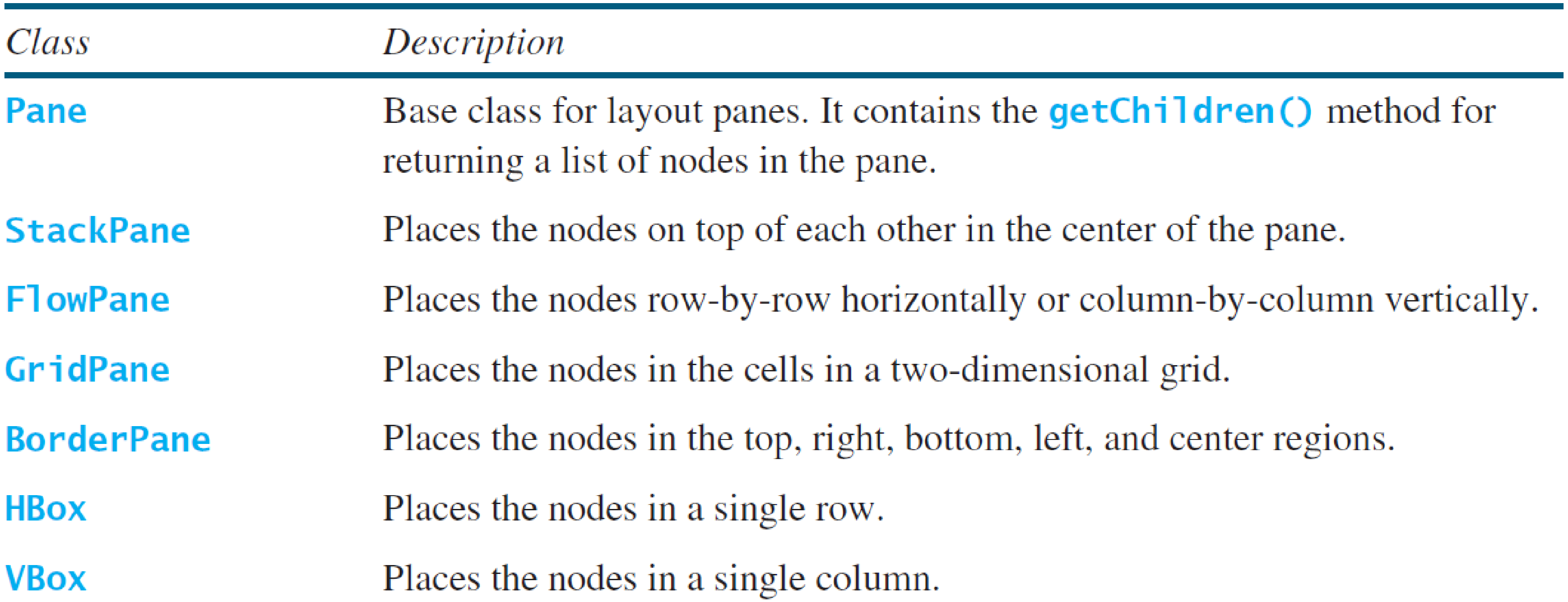
Stage
javafx.stage (window)

Scene
javafx.scene

Scene graph

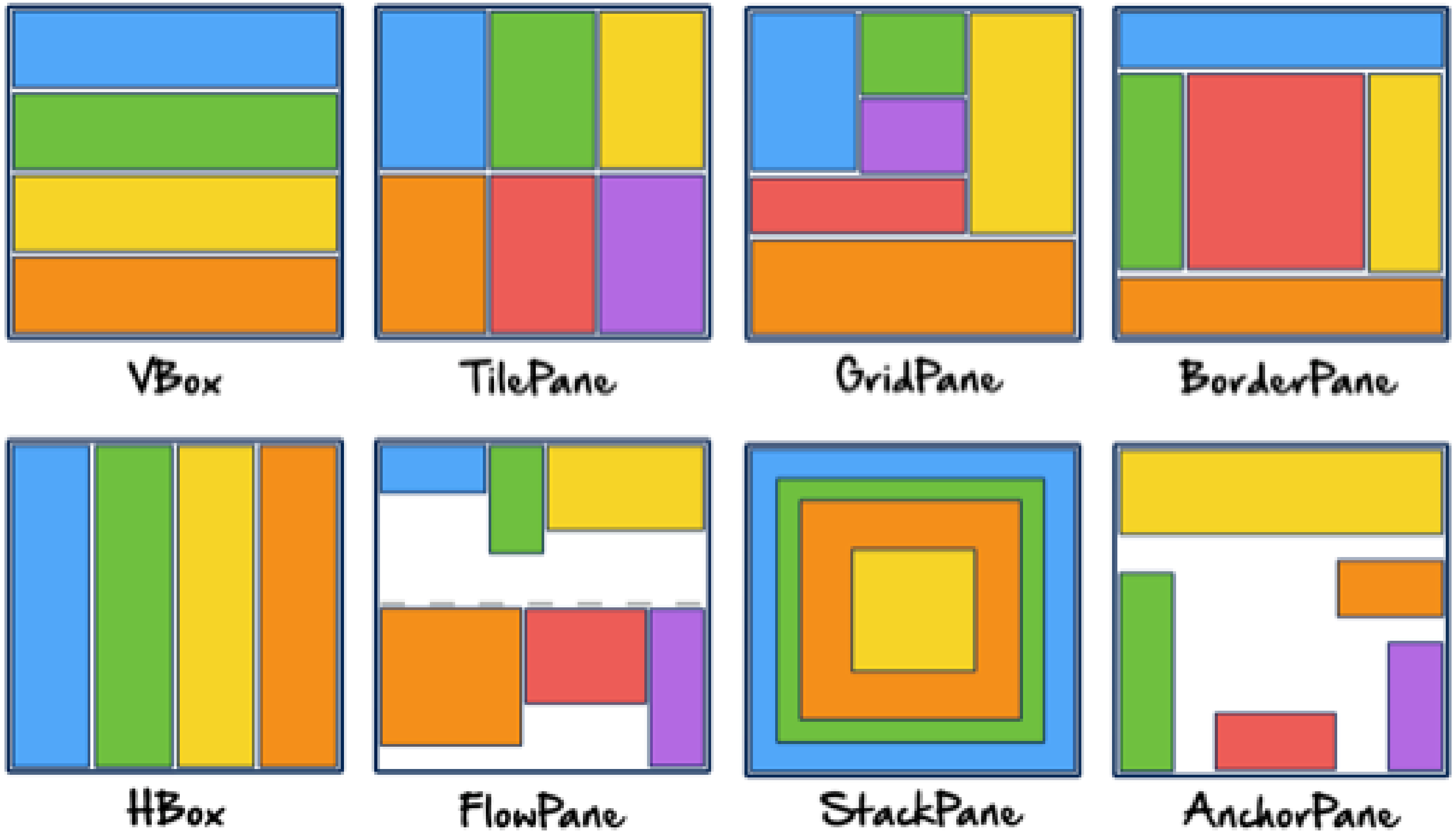
Layout Pane

- JavaFX provides many types of panes for organizing nodes in a container.



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.

Layout Pane (cont.)



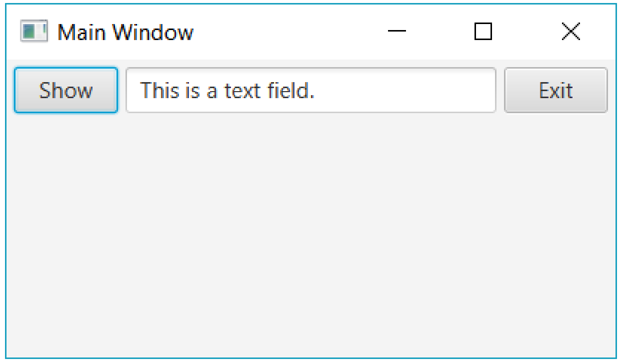
Reference: https://dzone.com/refcardz/javafx-8-1

Examples

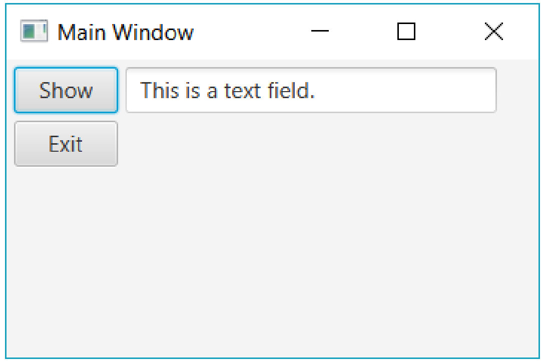
MainWindow.java

```java
package application;

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

public class MainWindow extends Application {
    @Override
    public void start(Stage primaryStage) {
    // create the flow pane as root node
    FlowPane root = new FlowPane();
    root.setPadding(new Insets(5));
    root.setHgap(5);
    root.setVgap(5);

        Button exitButton = new Button(" Exit ");
        exitButton.setPrefWidth(70);
        Button showButton = new Button(" Show ");
        showButton.setPrefWidth(70);

        TextField text = new TextField("This is a
                           text field.");
        text.setPrefWidth(250);
```

```java
root.getChildren().addAll(showButton,text,exitBu
tton);

        Scene scene = new Scene(root, 410, 200);

        primaryStage.setTitle("Main Window");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

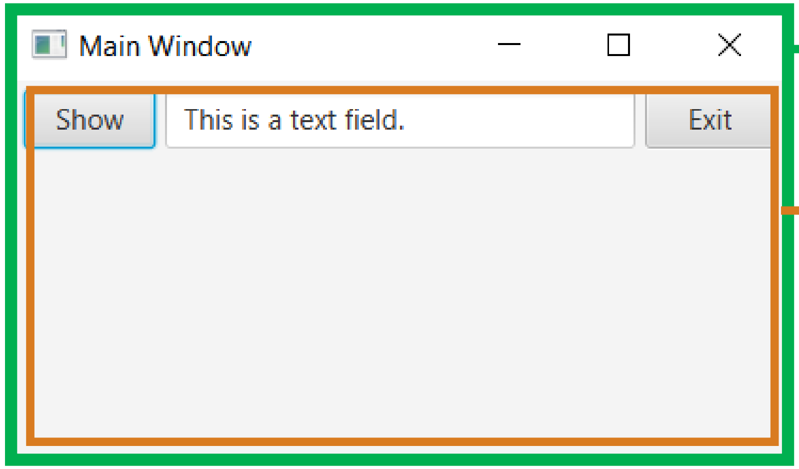


This is a text field.



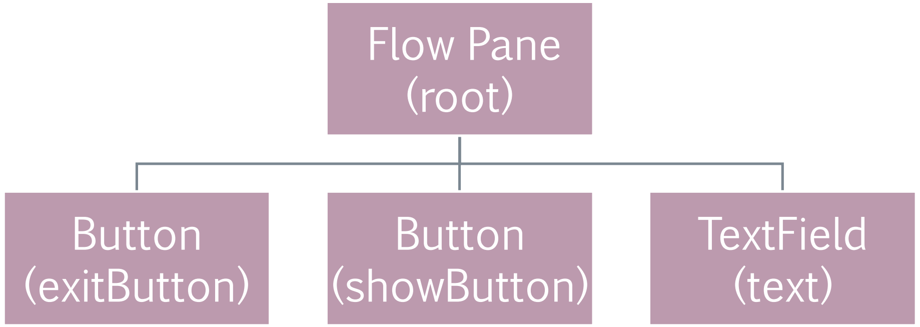
This is a text field.

Examples (cont.)



Stage
javafx.stage (window)

Scene
javafx.scene

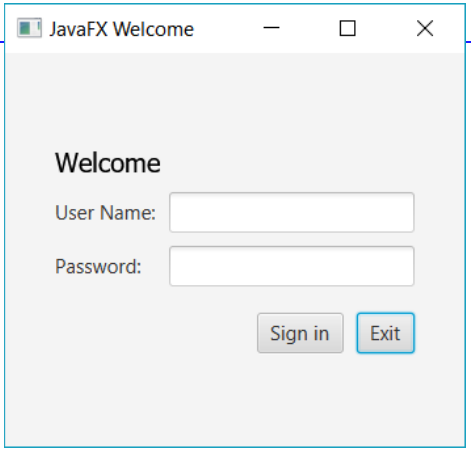


Scene graph

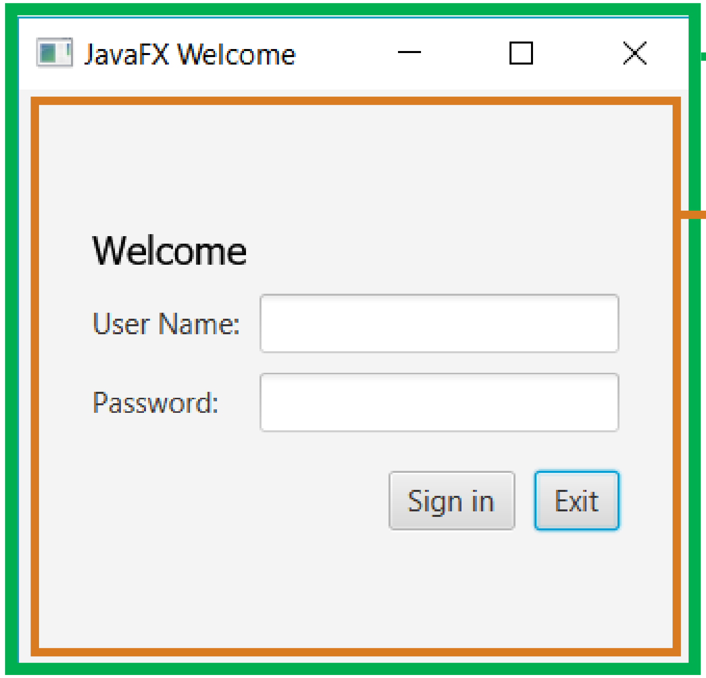
Examples (cont.)

Welcome.java

```
public class Welcome extends Application {

@Override
public void start(Stage primaryStage) {

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

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

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

      TextField userTextField = new TextField();
      grid.add(userTextField, 1, 1);

      Label pw = new Label("Password:");
      grid.add(pw, 0, 2);

      PasswordField pwBox = new PasswordField();
      grid.add(pwBox, 1, 2);

      HBox hbBtn = new HBox(10);
      hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
      Button signinBtn = new Button("Sign in");
      Button exitBtn = new Button("Exit");
      hbBtn.getChildren().addAll(signinBtn,exitBtn);
      grid.add(hbBtn, 1, 4);

      Scene scene = new Scene(grid, 350, 300);

      primaryStage.setScene(scene);
      primaryStage.setTitle("JavaFX Welcome");
      primaryStage.show();
      }

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



Examples (cont.)



JavaFX Welcome

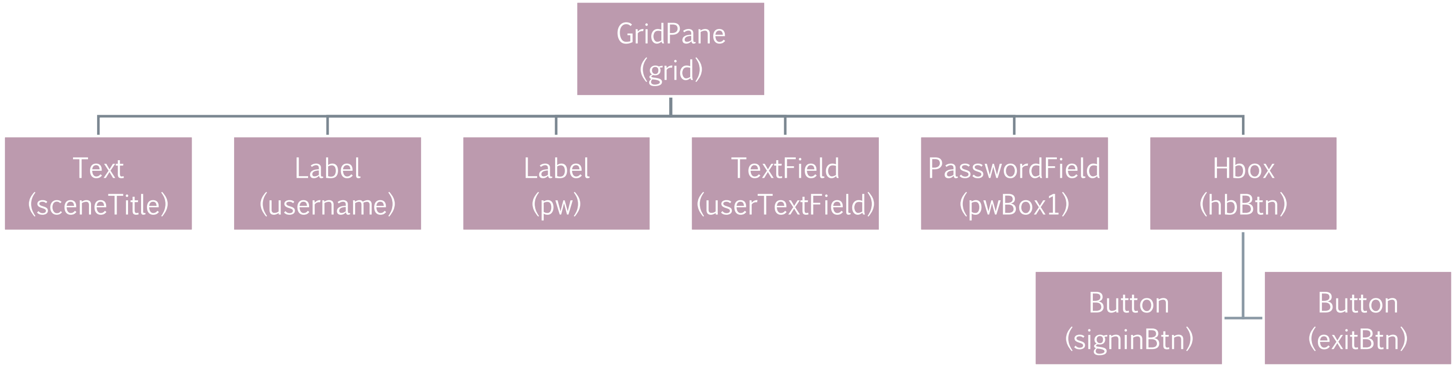
Welcome

User Name:

Password:

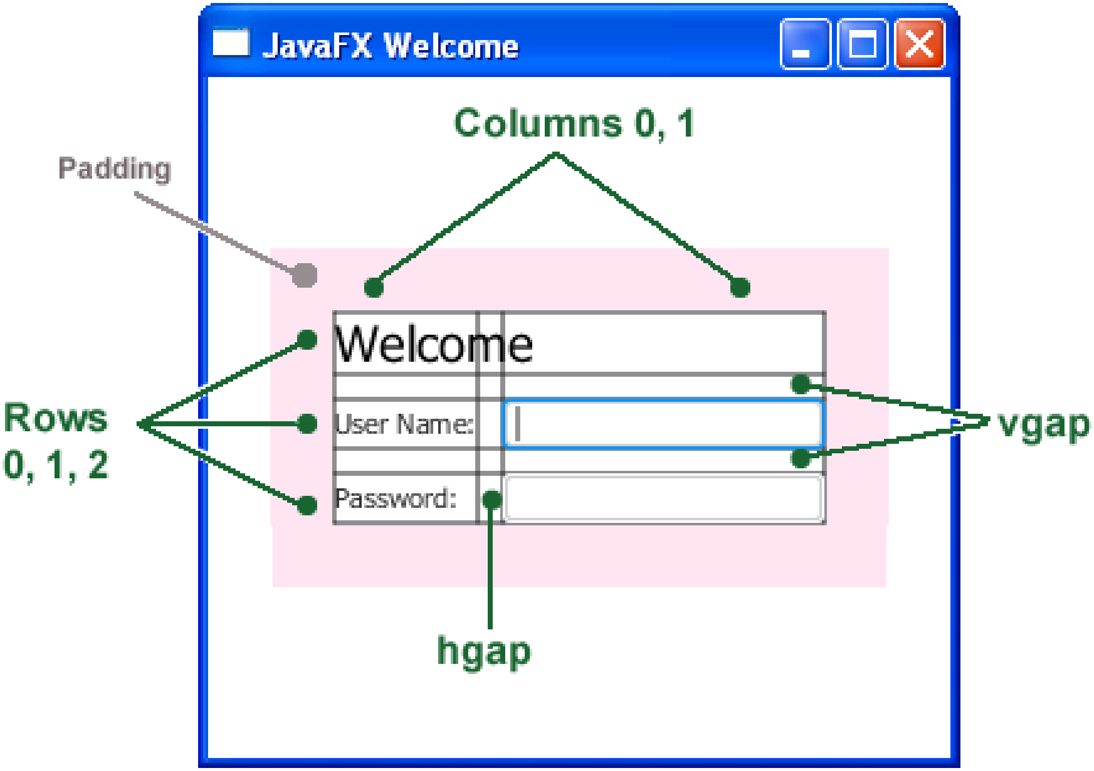
Stage
javafx.stage (window)

Scene
javafx.scene



Scene graph

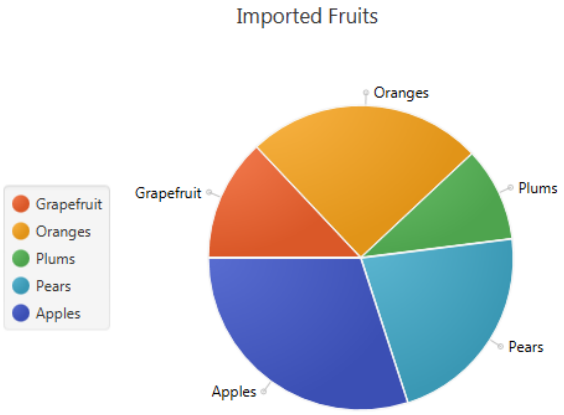
Examples (cont.)

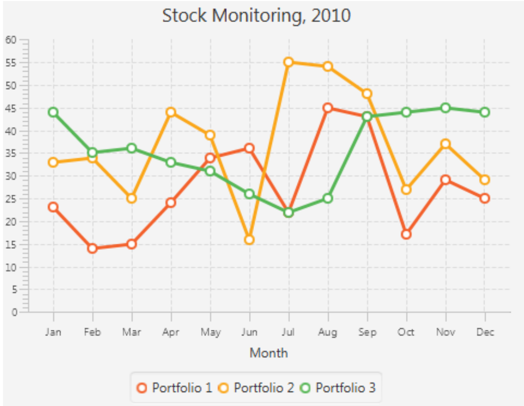


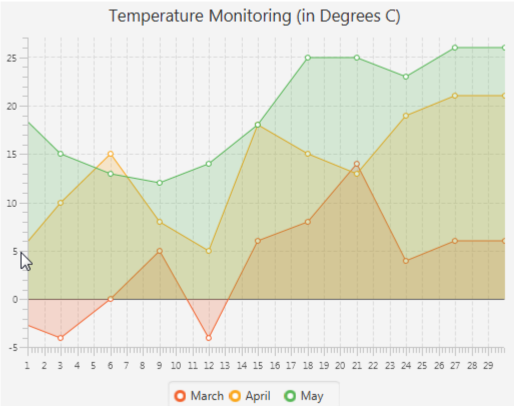
Reference: http://docs.oracle.com/javafx/2/get_started/form.htm

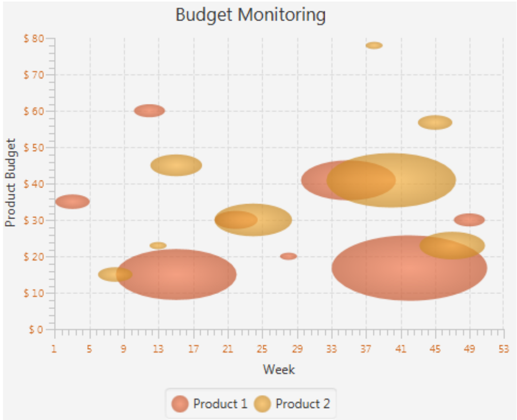
Charts

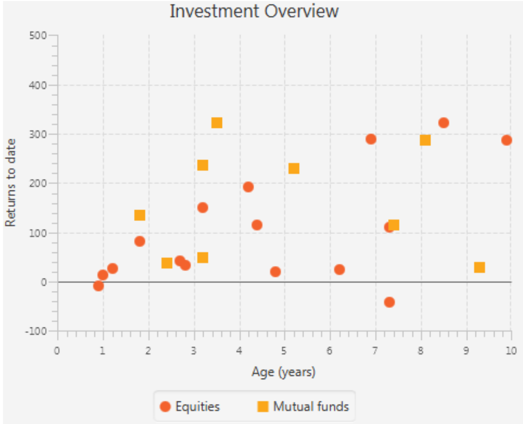
- javafx.scene.chart package

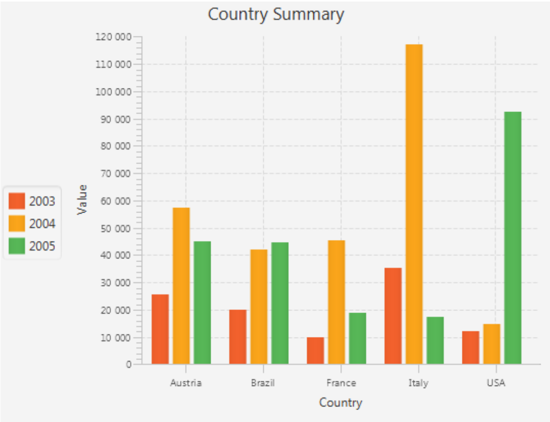




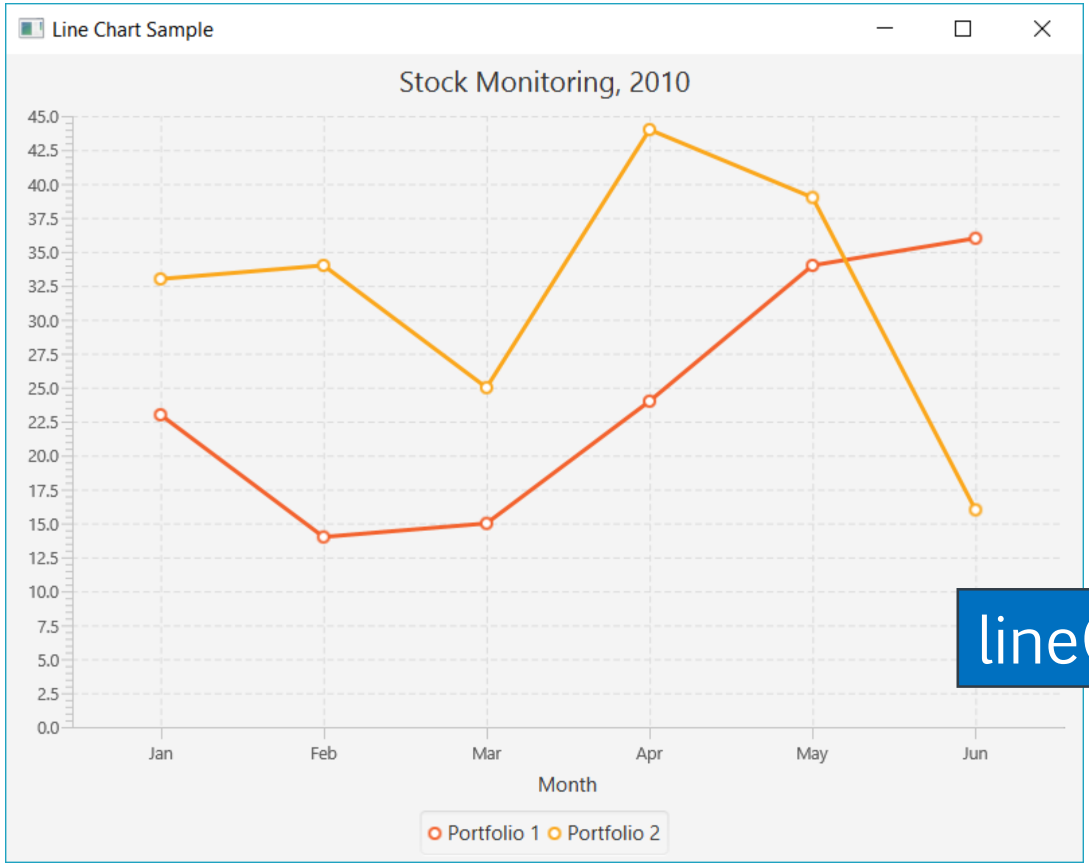








Charts (cont.)



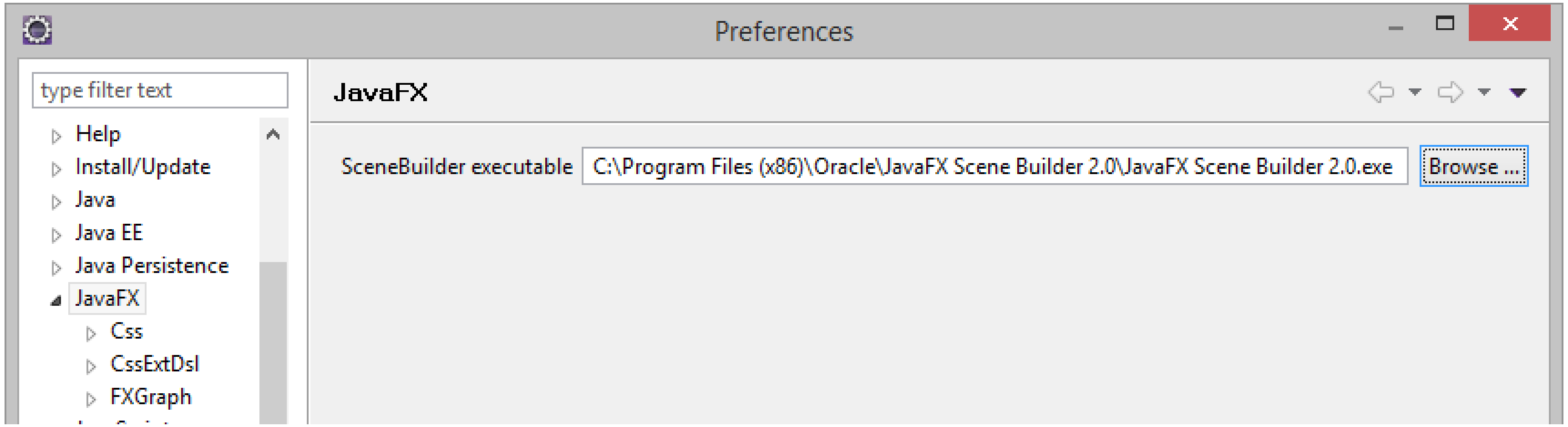
lineChartSample.java

Scene builder

- JavaFX Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding.
- FXML code for the layout that they are creating is automatically generated in the background.
- FXML file that can then be combined with a Java project by binding the UI to the application's logic

Scene builder (cont.)

- How to install JavaFX Scene Builder
  - Install E(fx)clipse into Eclipse http://o7planning.org/en/10619/install-efxclipse-into-eclipse
  - Download JavaFX Scene Builder http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-1x-archive-2199384.html
  - Configuring Eclipse to use the Scene Builder
    Window > Preferences



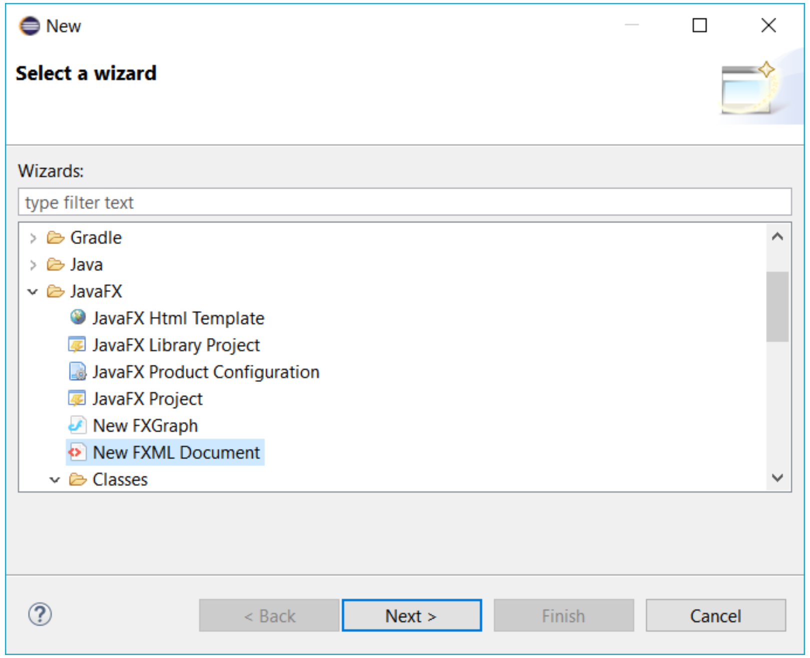
Preferences

JavaFX

SceneBuilder executable: C:\Program Files (x86)\Oracle\JavaFX Scene Builder 2.0\JavaFX Scene Builder 2.0.exe

Scene builder (cont.)

- New > other > New FXML Document



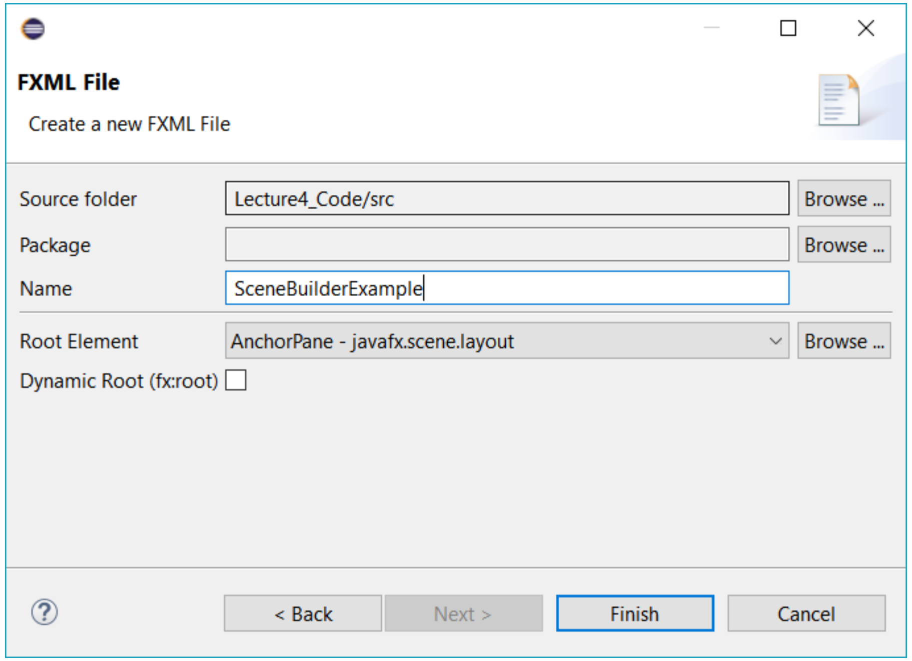
New

Select a wizard

Wizards:

type filter text

- Gradle
- Java
- JavaFX
  - JavaFX Html Template
  - JavaFX Library Project
  - JavaFX Product Configuration
  - JavaFX Project
  - New FXGraph
  - New FXML Document
  - Classes



FXML File

Create a new FXML File

Source folder: Lecture4_Code/src

Package:

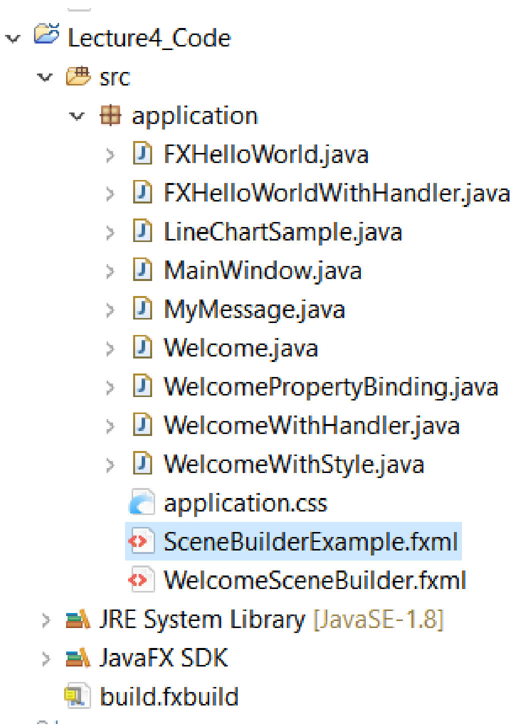
Name: SceneBuilderExample

Root Element: AnchorPane - javafx.scene.layout

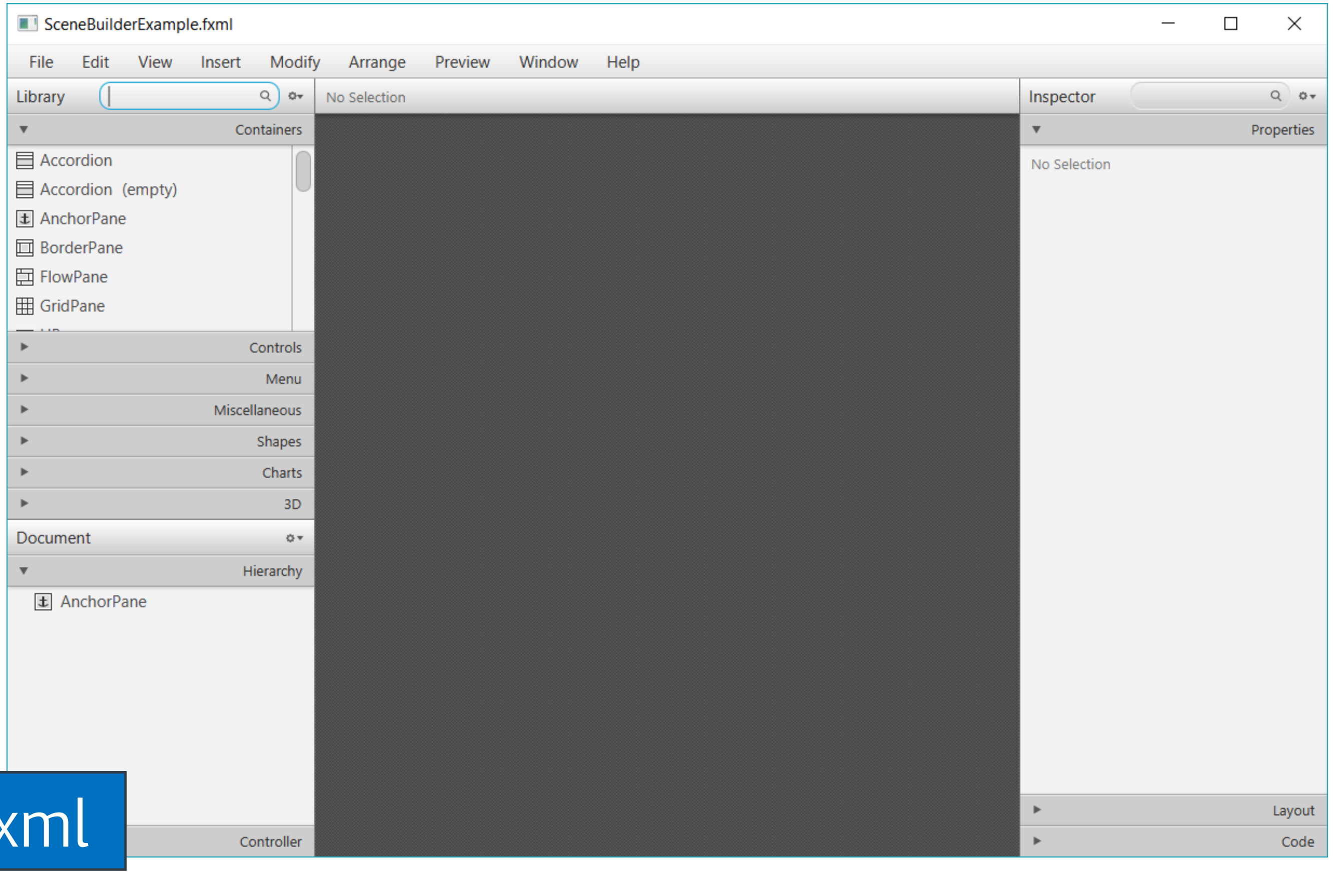
Dynamic Root (fx:root) □

Scene builder (cont.)

› Right click .fxml file > open with SceneBuilder

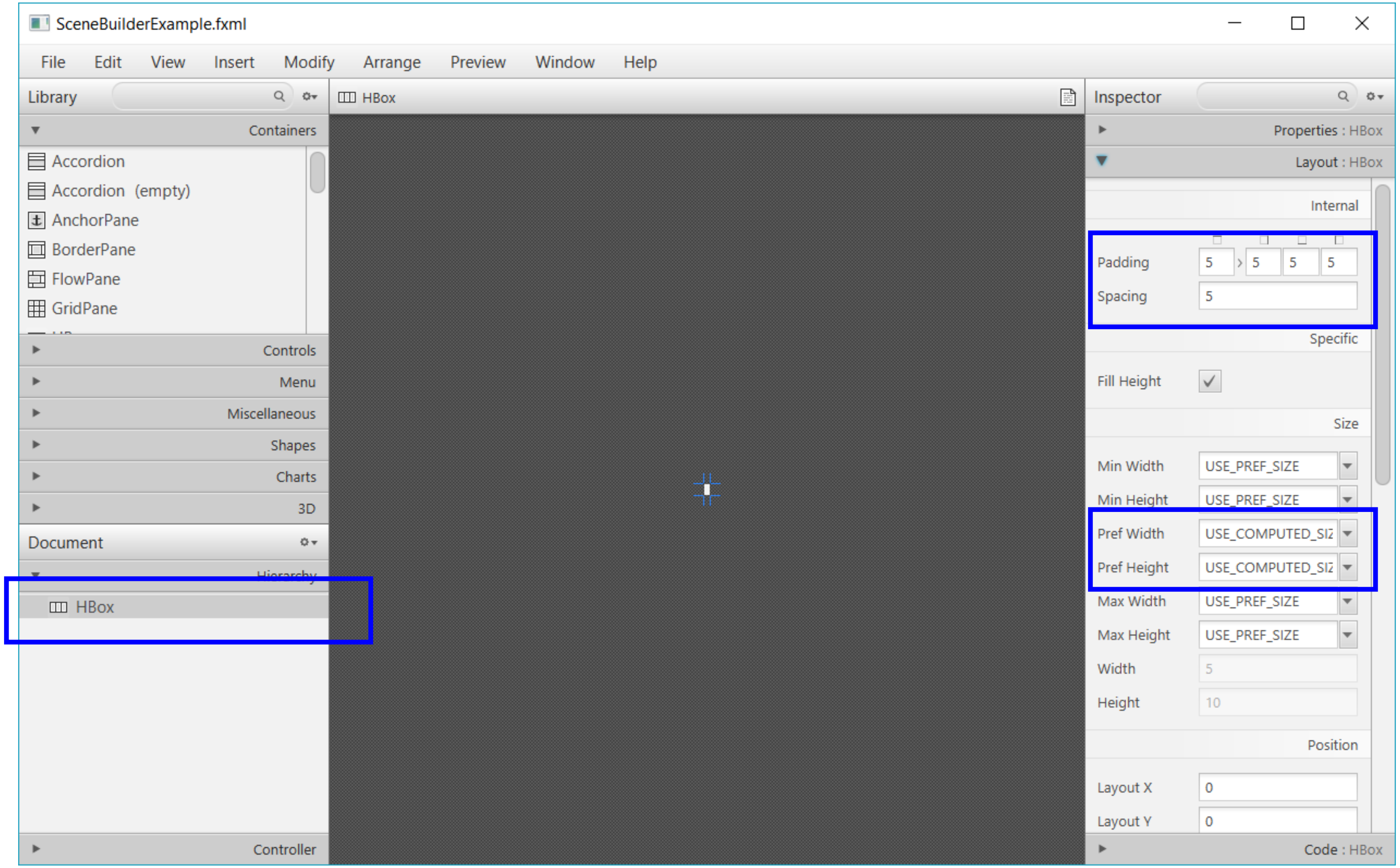


```
v Lecture4_Code
  v src
    v application
      > FXHelloWorld.java
      > FXHelloWorldWithHandler.java
      > LineChartSample.java
      > MainWindow.java
      > MyMessage.java
      > Welcome.java
      > WelcomePropertyBinding.java
      > WelcomeWithHandler.java
      > WelcomeWithStyle.java
        application.css
        SceneBuilderExample.fxml
        WelcomeSceneBuilder.fxml
  > JRE System Library [JavaSE-1.8]
  > JavaFX SDK
    build.fxbuild
```

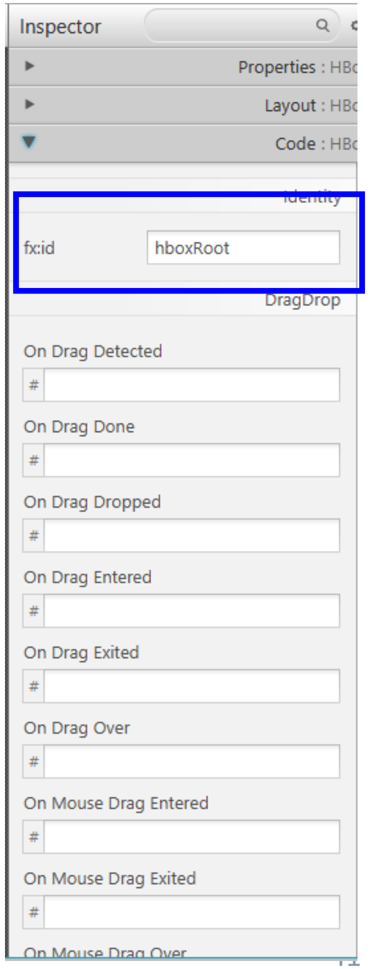


SceneBuilderExample.fxml

Scene builder (cont.)



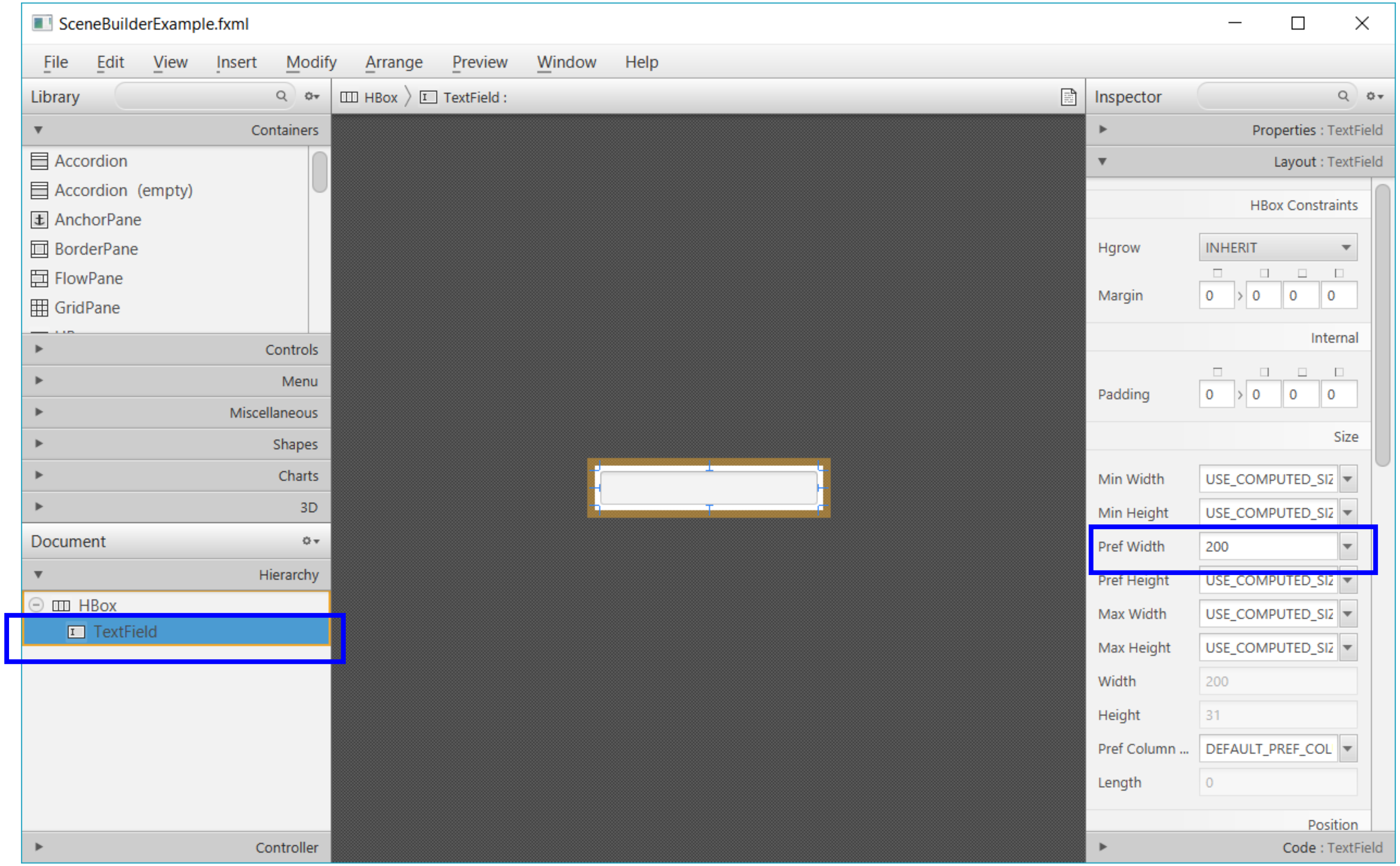
Padding 5 5 5 5
Spacing 5
Fill Height ✓
Min Width USE_PREF_SIZE
Min Height USE_PREF_SIZE
Pref Width USE_COMPUTED_SIZ
Pref Height USE_COMPUTED_SIZ
Max Width USE_PREF_SIZE
Max Height USE_PREF_SIZE
Width 5
Height 10
Layout X 0
Layout Y 0



fx:id hboxRoot

On Drag Detected #
On Drag Done #
On Drag Dropped #
On Drag Entered #
On Drag Exited #
On Drag Over #
On Mouse Drag Entered #
On Mouse Drag Exited #
On Mouse Drag Over

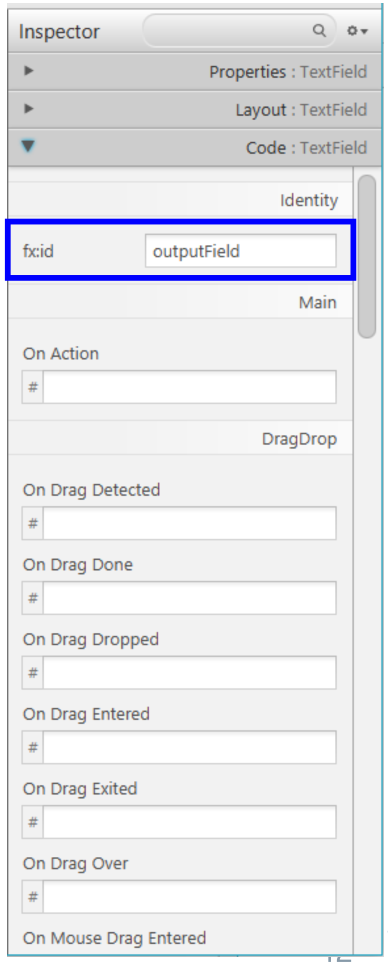
Scene builder (cont.)



SceneBuilderExample.fxml

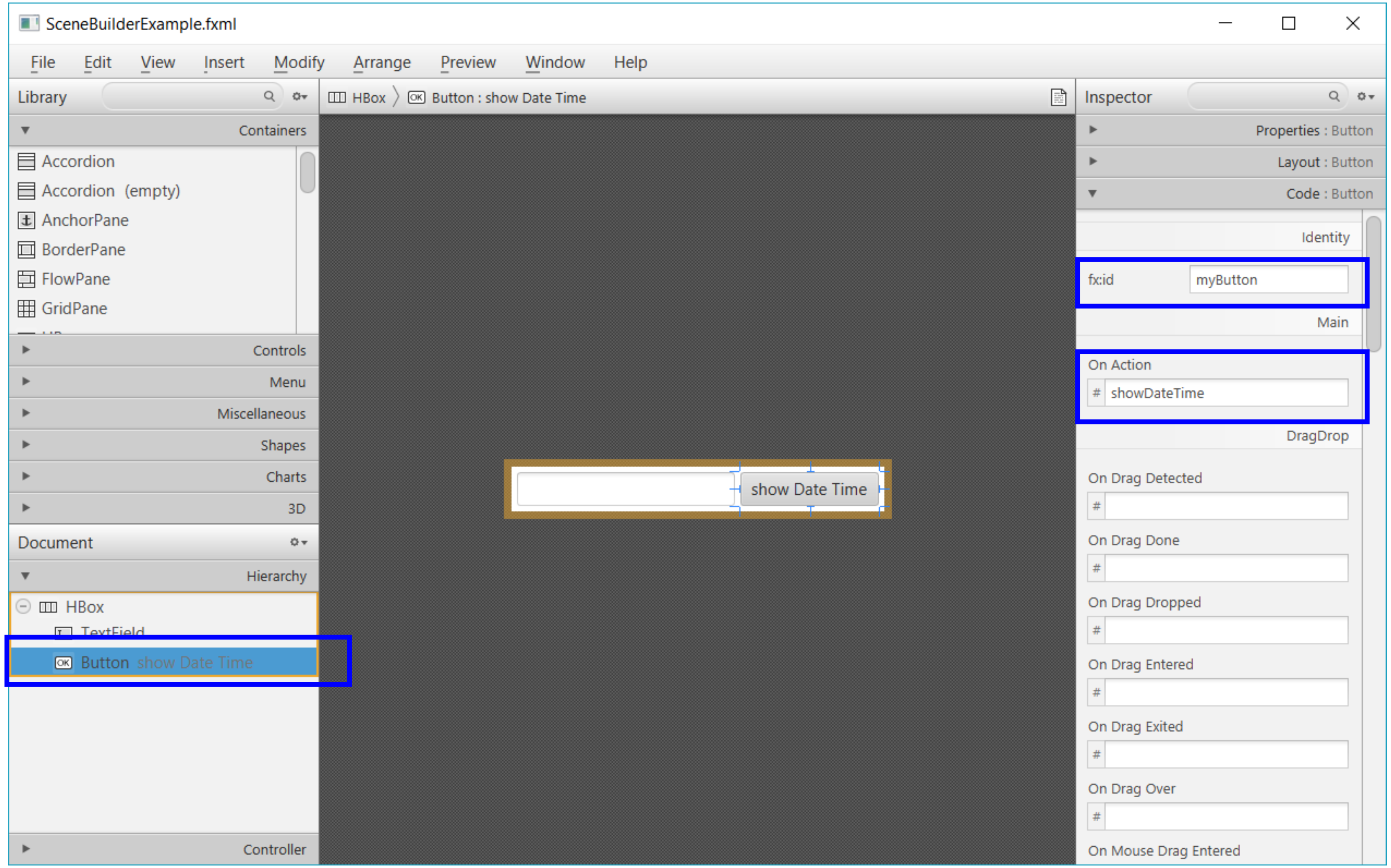
Hierarchy: HBox > TextField

Pref Width: 200



fx:id: outputField

Scene builder (cont.)



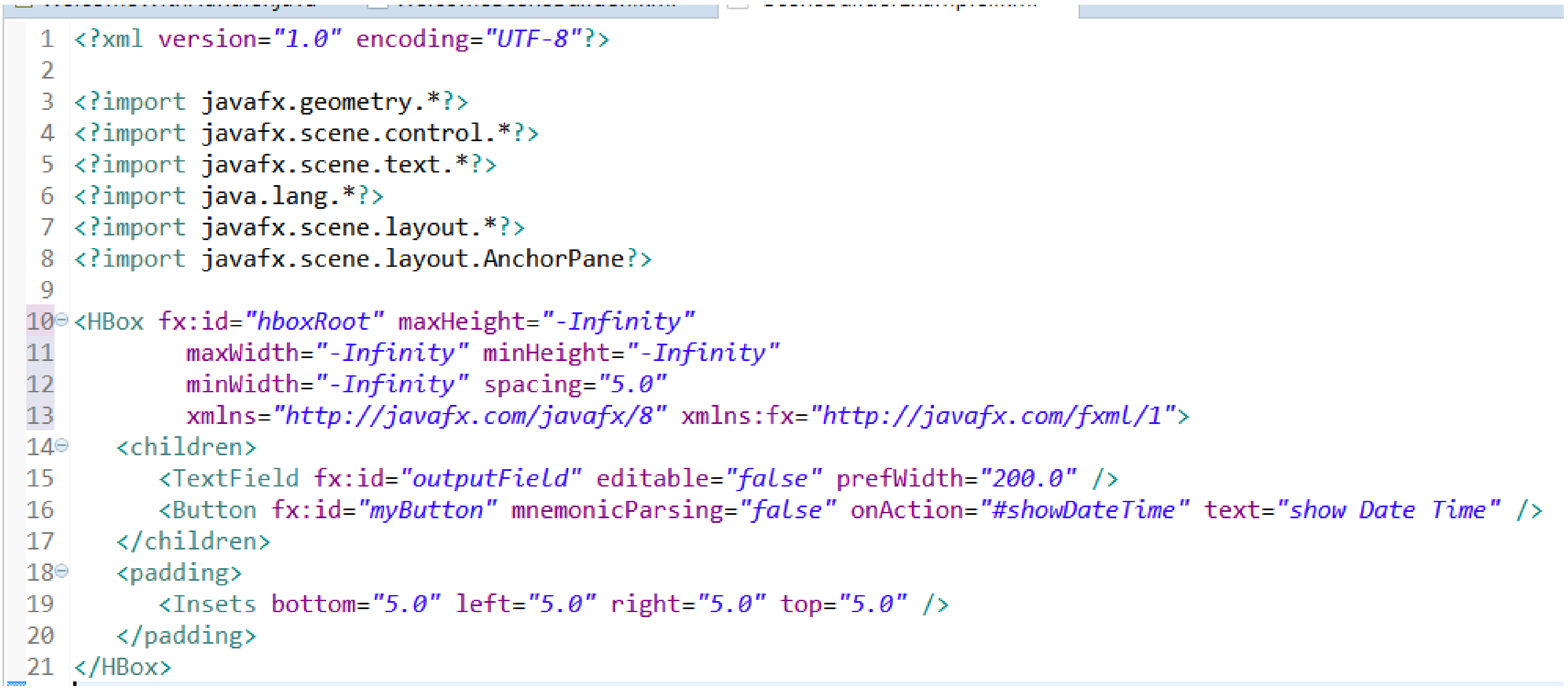
SceneBuilderExample.fxml

fx:id: myButton

On Action: # showDateTime

Scene builder (cont.)

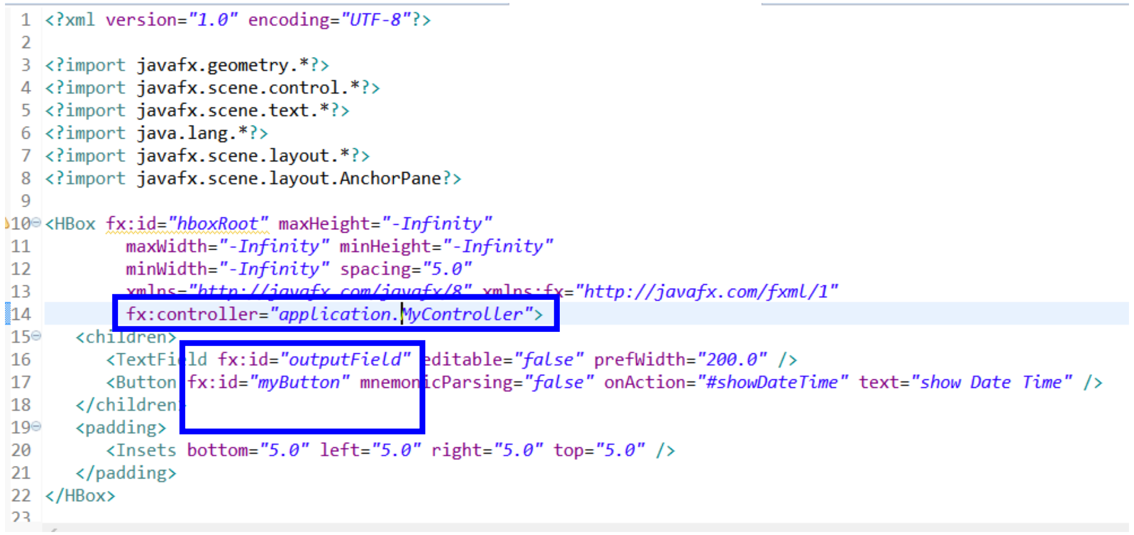
- Save
- Double click file in eclipse to view FXML



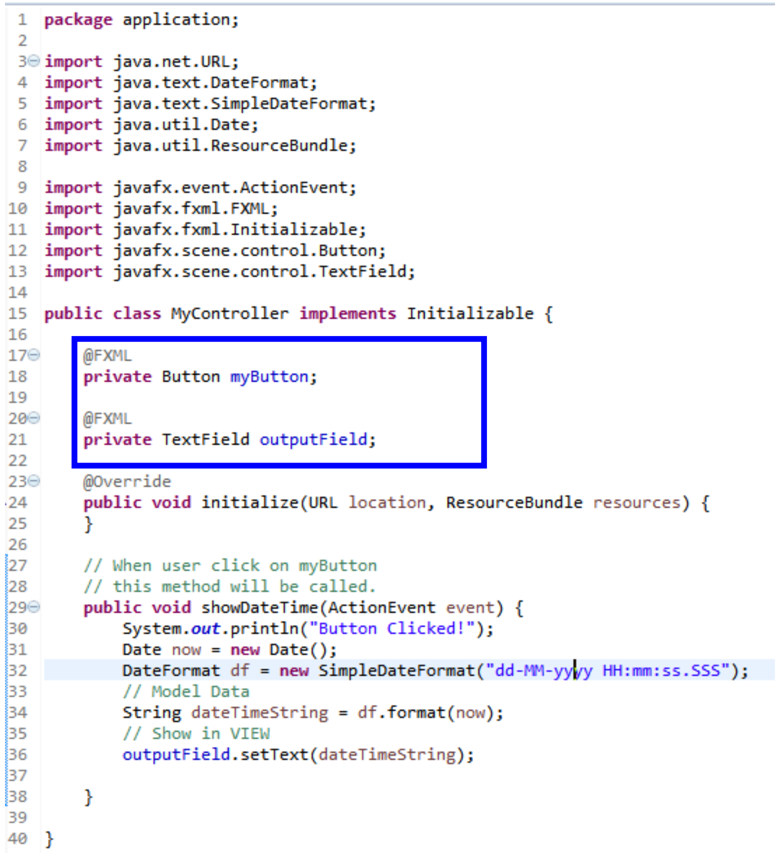
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>

<HBox fx:id="hboxRoot" maxHeight="-Infinity"
        maxWidth="-Infinity" minHeight="-Infinity"
        minWidth="-Infinity" spacing="5.0"
        xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <TextField fx:id="outputField" editable="false" prefWidth="200.0" />
      <Button fx:id="myButton" mnemonicParsing="false" onAction="#showDateTime" text="show Date Time" />
   </children>
   <padding>
      <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
   </padding>
</HBox>
```

Scene builder (cont.)

- Adding the attribute fx:controller to <Hbox>, the Controller will be useful to the Controls lying inside Hbox such as myButton and outputField.



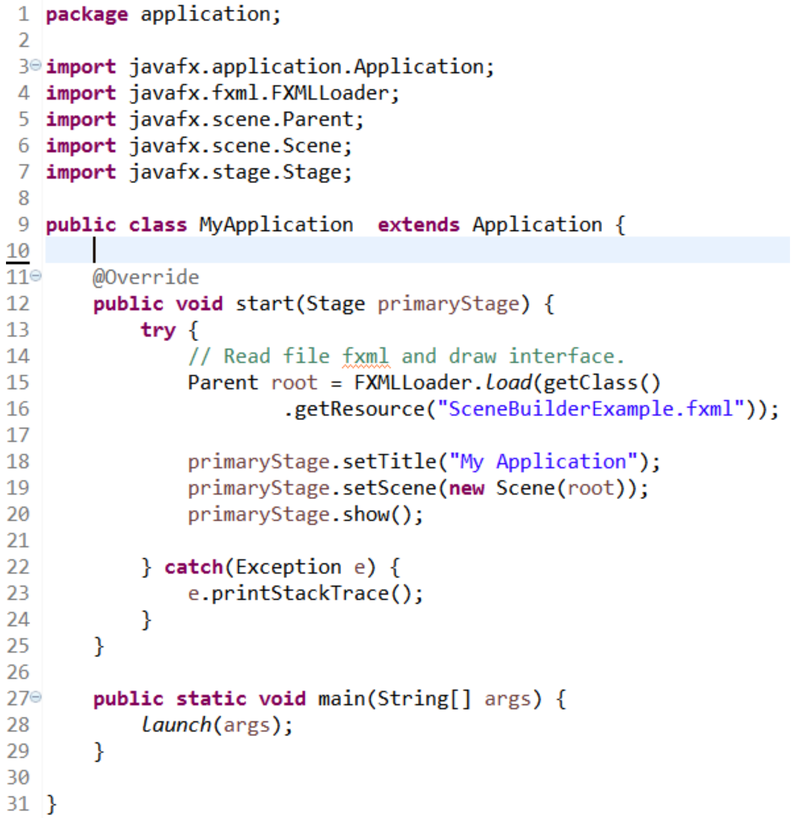
```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>

<HBox fx:id="hboxRoot" maxHeight="-Infinity"
        maxWidth="-Infinity" minHeight="-Infinity"
        minWidth="-Infinity" spacing="5.0"
        xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
        fx:controller="application.MyController">
    <children>
        <TextField fx:id="outputField" editable="false" prefWidth="200.0" />
        <Button fx:id="myButton" mnemonicParsing="false" onAction="#showDateTime" text="show Date Time" />
    </children>
    <padding>
        <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
    </padding>
</HBox>
```



```java
package application;

import java.net.URL;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.ResourceBundle;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class MyController implements Initializable {

    @FXML
    private Button myButton;

    @FXML
    private TextField outputField;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
    }

    // When user click on myButton
    // this method will be called.
    public void showDateTime(ActionEvent event) {
        System.out.println("Button Clicked!");
        Date now = new Date();
        DateFormat df = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss.SSS");
        // Model Data
        String dateTimeString = df.format(now);
        // Show in VIEW
        outputField.setText(dateTimeString);

    }

}
```

Scene builder (cont.)

› Run “MyApplication”

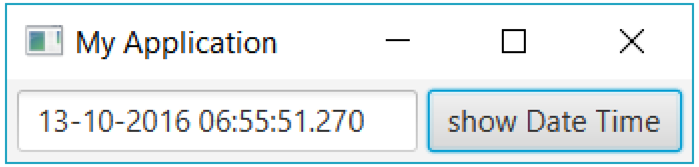


```
package application;

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

public class MyApplication  extends Application {

    @Override
    public void start(Stage primaryStage) {
        try {
            // Read file fxml and draw interface.
            Parent root = FXMLLoader.load(getClass()
                    .getResource("SceneBuilderExample.fxml"));

            primaryStage.setTitle("My Application");
            primaryStage.setScene(new Scene(root));
            primaryStage.show();

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

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

}
```

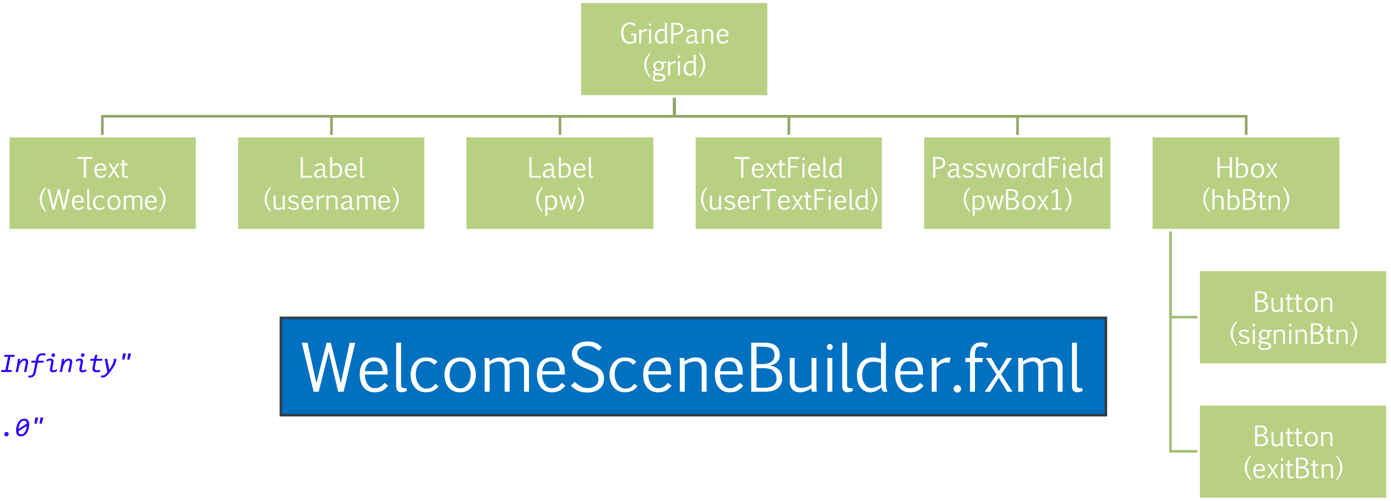
SceneBuilderExample.fxml

MyController.java

MyApplication.java



FXML



WelcomeSceneBuilder.fxml

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

<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>

<GridPane hgap="10.0" maxHeight="-Infinity" maxWidth="-Infinity"
          minHeight="-Infinity" minWidth="-Infinity"
          prefHeight="300.0" prefWidth="350.0" vgap="10.0"
          xmlns="http://javafx.com/javafx/8"
          xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Welcome">
         <font>
            <Font name="Tahoma" size="20.0" />
         </font>
      </Text>
      <Label text="User Name:" GridPane.rowIndex="1" />
      <Label text="Password:" GridPane.rowIndex="2" />
      <HBox alignment="BOTTOM_RIGHT" prefHeight="100.0"
            prefWidth="200.0" spacing="10.0" GridPane.columnIndex="1"
            GridPane.rowIndex="4">
         <children>
            <Button mnemonicParsing="false" text="Sign in" />
            <Button mnemonicParsing="false" text="Exit" />
         </children>
      </HBox>
      <TextField GridPane.columnIndex="1" GridPane.rowIndex="1" />
      <PasswordField GridPane.columnIndex="1" GridPane.rowIndex="2" />
   </children>
```

```
   <columnConstraints>
      <ColumnConstraints hgrow="SOMETIMES" maxWidth="263.0"
                         minWidth="10.0" prefWidth="87.0" />
      <ColumnConstraints hgrow="SOMETIMES" maxWidth="463.0"
                         minWidth="10.0" prefWidth="203.0" />
   </columnConstraints>
   <padding>
      <Insets bottom="25.0" left="25.0" right="25.0" top="25.0" />
   </padding>
   <rowConstraints>
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
   </rowConstraints>
</GridPane>
```

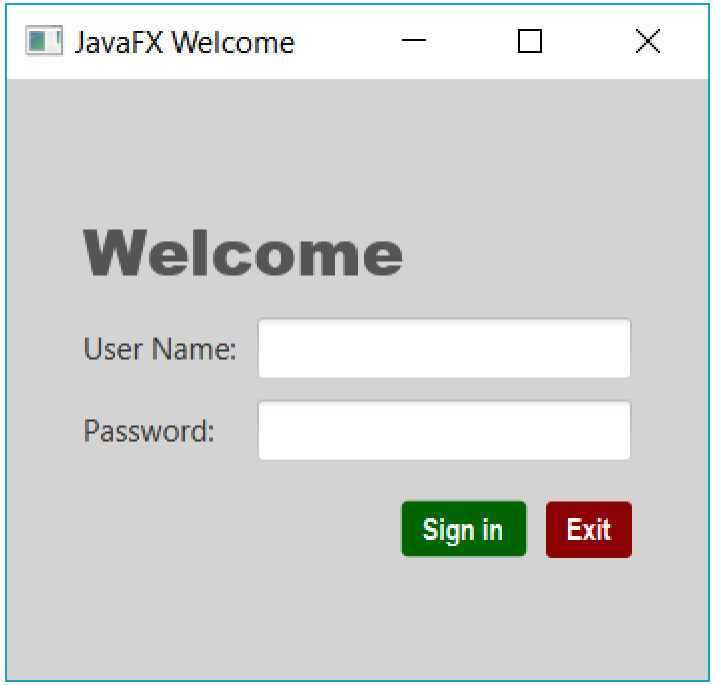
CSS

- JavaFX provides styling by Cascading Style Sheets(CSS).
- CSS support is based on the W3C CSS version 2.1
- JavaFX CSS document: http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

CSS (cont.)

WelcomeWithStyle.java

```
// set style
grid.setStyle("-fx-background-color:lightgray;");
scenetitle.setStyle("-fx-font-size: 32px;
                   -fx-font-family:\"Arial Black\";
                   -fx-fill: #555;");
signinBtn.setStyle("-fx-text-fill: white;
                   -fx-font-weight: bold;
                   -fx-font-family: \"Arial Narrow\";
                   -fx-background-color: darkgreen;");
exitBtn.setStyle("-fx-text-fill: white;
                   -fx-font-weight: bold;
                   -fx-font-family: \"Arial Narrow\";
                   -fx-background-color: darkred;");
```



Remarks: you can set same style for more than one node using “css class” or writing the style in separated file (not covered in this class)

Binding properties

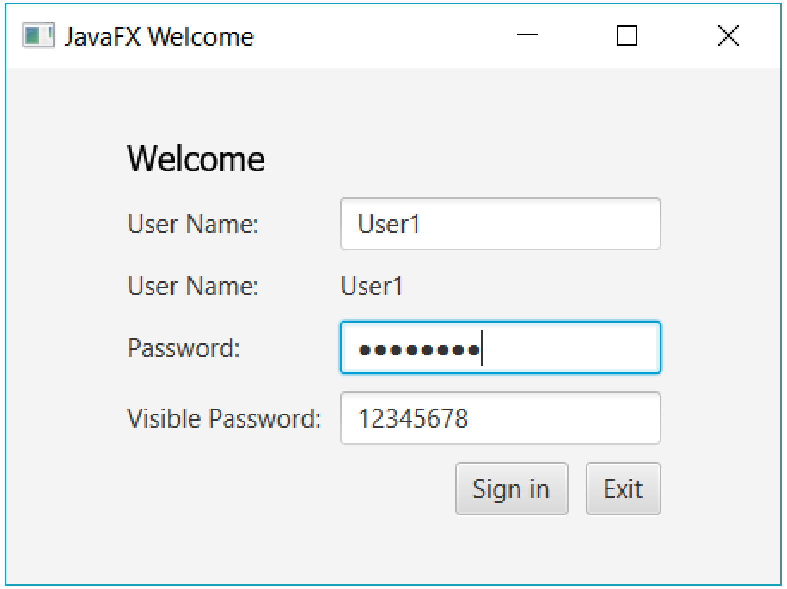
- JavaFX introduces a new concept called binding property
- Enables a target object to be bound to a source object.
- If the value in the source object changes, the target property is also changed automatically.
- The target object is simply called a binding object or a binding property.

Binding Properties (cont.)

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

Label userName1 = new Label("User Name:");
grid.add(userName1, 0, 2);
Label userNameOut = new Label();
grid.add(userNameOut, 1, 2);

// Unidirectional bindings
userNameOut.textProperty().bind(userTextField.textProperty());
```

```
Label pw1 = new Label("Password:");
grid.add(pw1, 0, 3);
PasswordField pwBox1 = new PasswordField();
grid.add(pwBox1, 1, 3);

Label pw2 = new Label("Visible Password:");
grid.add(pw2, 0, 4);
TextField pwBox2 = new TextField();
grid.add(pwBox2, 1, 4);

// Bidirectional bindings
pwBox1.textProperty().bindBidirectional(pwBox2.textProperty());
```

WelcomePropertyBinding.java



JavaFX Welcome

Welcome

User Name: User1

User Name: User1

Password: ••••••••

Visible Password: 12345678

Event Handling

- To make the program response to an action, you need to create a listener object that waits for a particular event to handle and modified the correspondence method.
- There are many events on GUI:
  - ActionEvent, InputEvent, ScrollToEvent, WindowEvent, WebEvent, MouseEvent, KeyEvent, ...
- JavaFX event is an instance of the javafx.event.Event class or its subclass

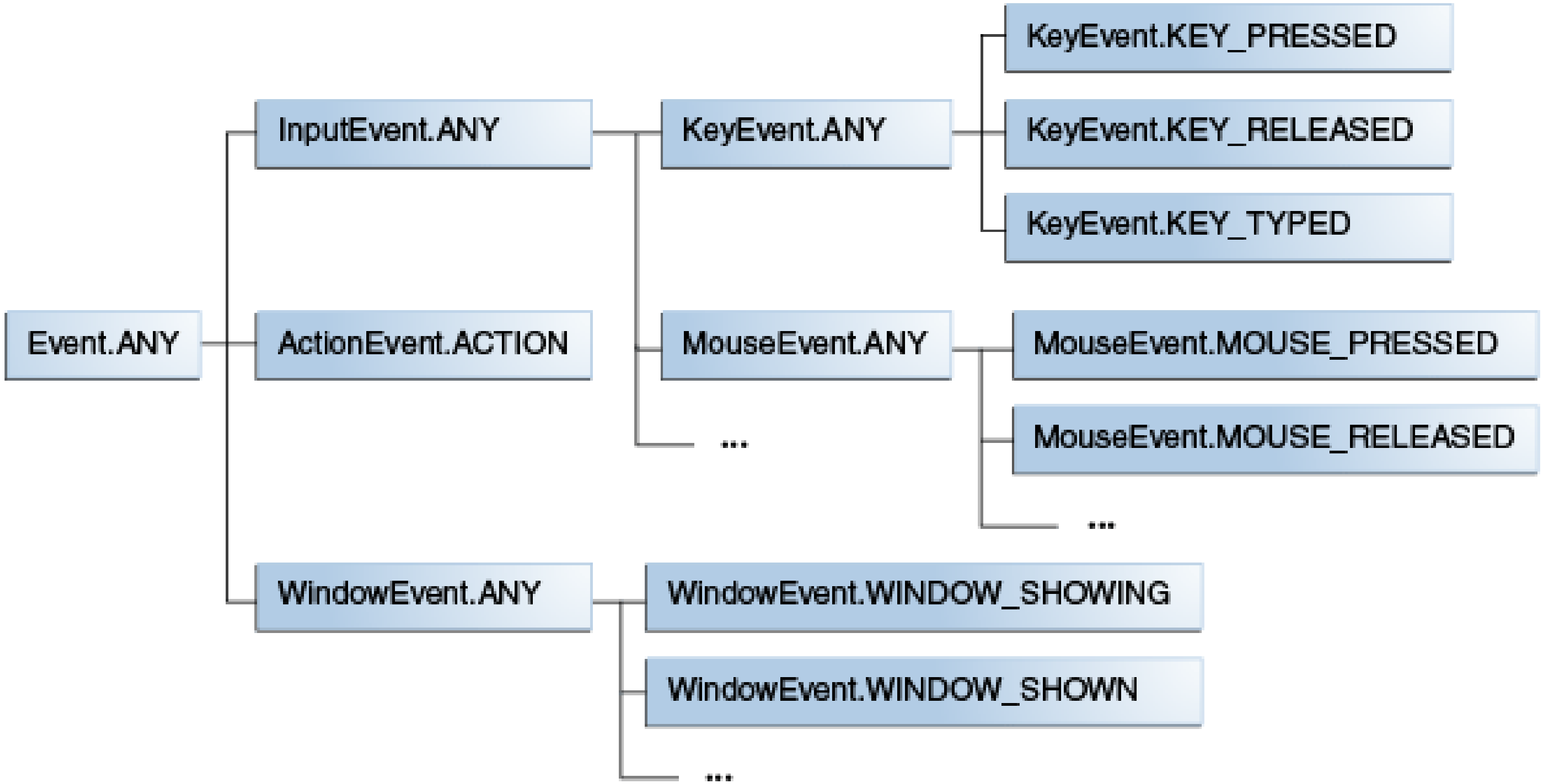
Event Handling

- Use the setOnXXX methods to register event handlers

```
setOnEvent-type(EventHandler<? super event-class>  value)
```

  - Event-type is the type of event that the handler processes,
    setOnKeyTyped for Key Typed events
    setOnMouseClicked for Mouse Clicked events.
  - event-class is the class that defines the event type,
    KeyEvent for events related to keyboard input
    MouseEvent for events related to mouse input.
- Override handle method

Event Handling (cont.)



Event type hierarchy
Reference: http://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

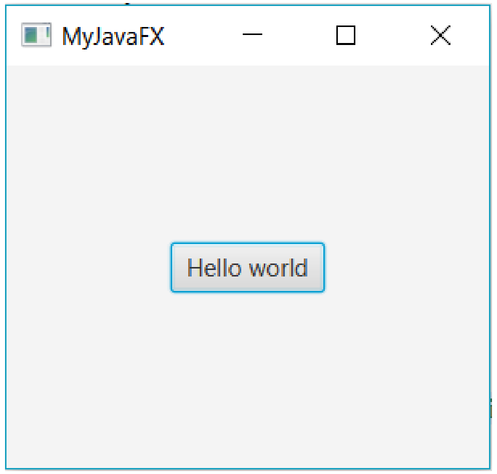
Event Handling (cont.)

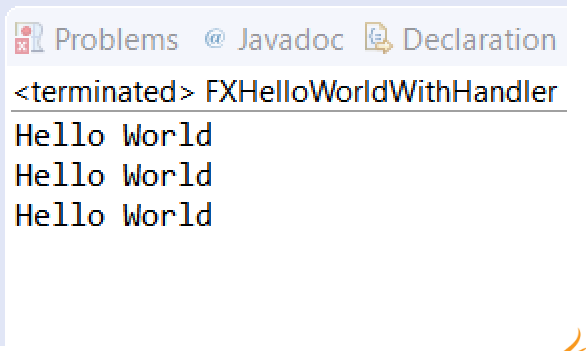
FXHelloWorldWithHandler.java

```java
package application;

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

public class FXHelloWorld extends Application {
      // Override the start method in the Application class
      @Override
      public void start(Stage primaryStage) {
            // Create a scene and place a button in the scene
            Button btn = new Button("Hello world");
            . . .
            StackPane root = new StackPane();
            root.getChildren().add(btn);
            Scene scene = new Scene(root, 300, 250);
            primaryStage.setTitle("MyJavaFX"); // Set the stage title
            primaryStage.setScene(scene); // Place the scene
            primaryStage.show();
      }

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

```java
import javafx.event.ActionEvent;

import javafx.event.EventHandler;
```

```java
// set event handler

btn.setOnAction(new EventHandler<ActionEvent>() {

      public void handle(ActionEvent event) {

            System.out.println("Hello World");

      }

});
```





```
<terminated> FXHelloWorldWithHandler
Hello World
Hello World
Hello World
```

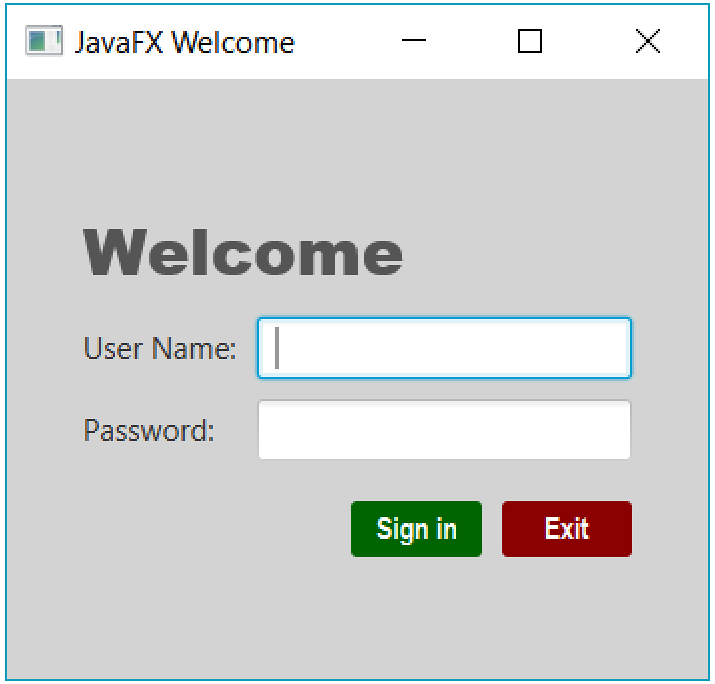
Event Handling (cont.)

- setOnAction() method is used to register an event handler.
- handle() method in the event handler is called when user clicks the button and it print "Hello World" to the console.

Event Handling (cont.)

- Clear User Name when press ESC
- Change button width if mouse is over
- Popup welcome dialog when click Sign in
- Close application when click Exit

WelcomeWithHandler.java


JavaFX Welcome

Welcome

User Name:

Password:

Common Event-Handling Problem

- A component does not generate the events it should.
  - Did you register the right kind of listener to detect the events?
  - Did you register the listener to the right object?
  - Did you implement the event handler correctly?

Export Jar

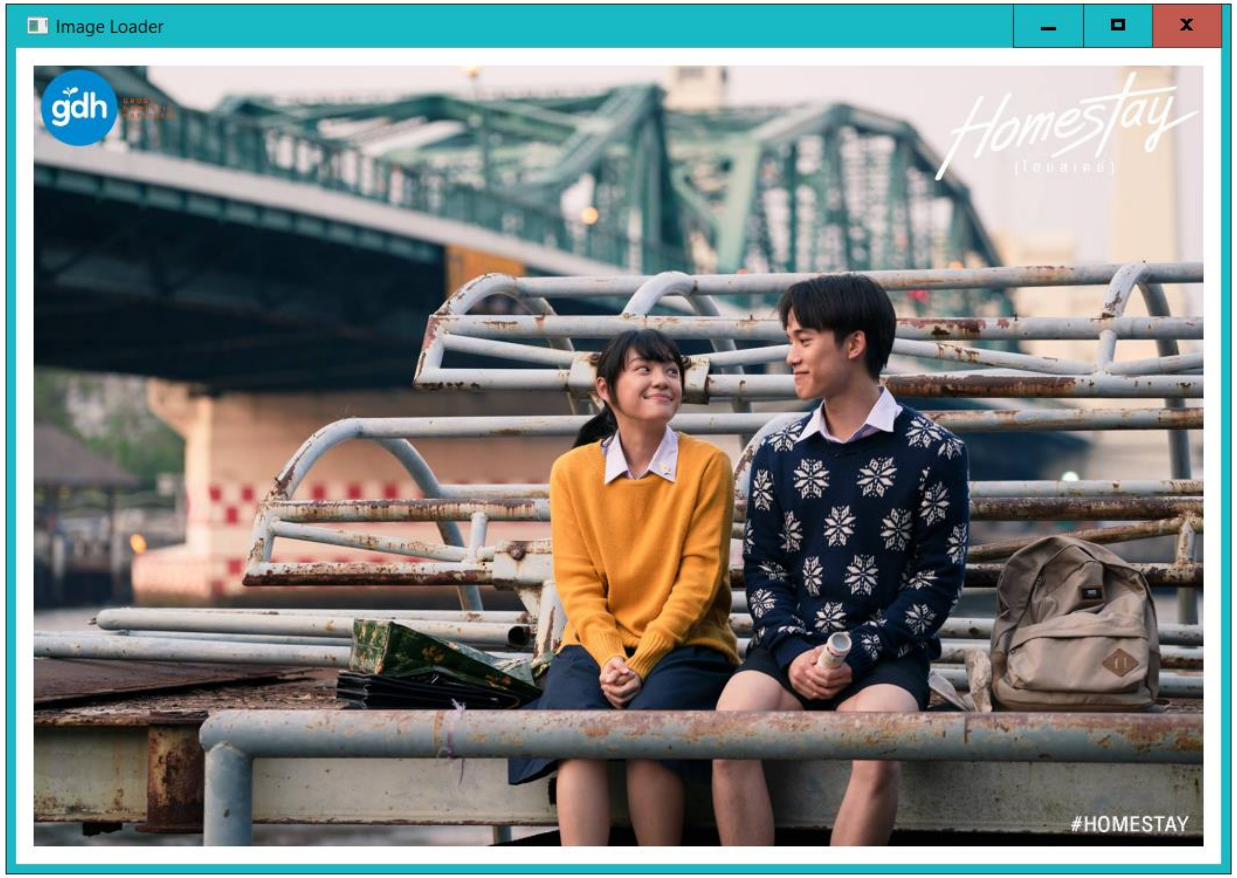
- We’ve managed to create our Java FX Application
- Let’s try out our application as an executable JAR

Export Jar (cont.)

- We've managed to create our Java FX Application
- Let's try out our application as an executable JAR

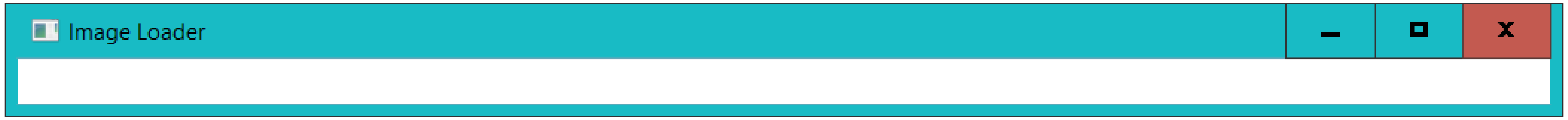
Export Jar (cont.)

- Run -> JAVA_FX_Image/ImageLoader.jar



Export Jar (cont.)

- Let copy our ImageLoader.jar to somewhere
- Run -> JAVA_FX_Image/Test_Jar/1_only_jar/run.jar



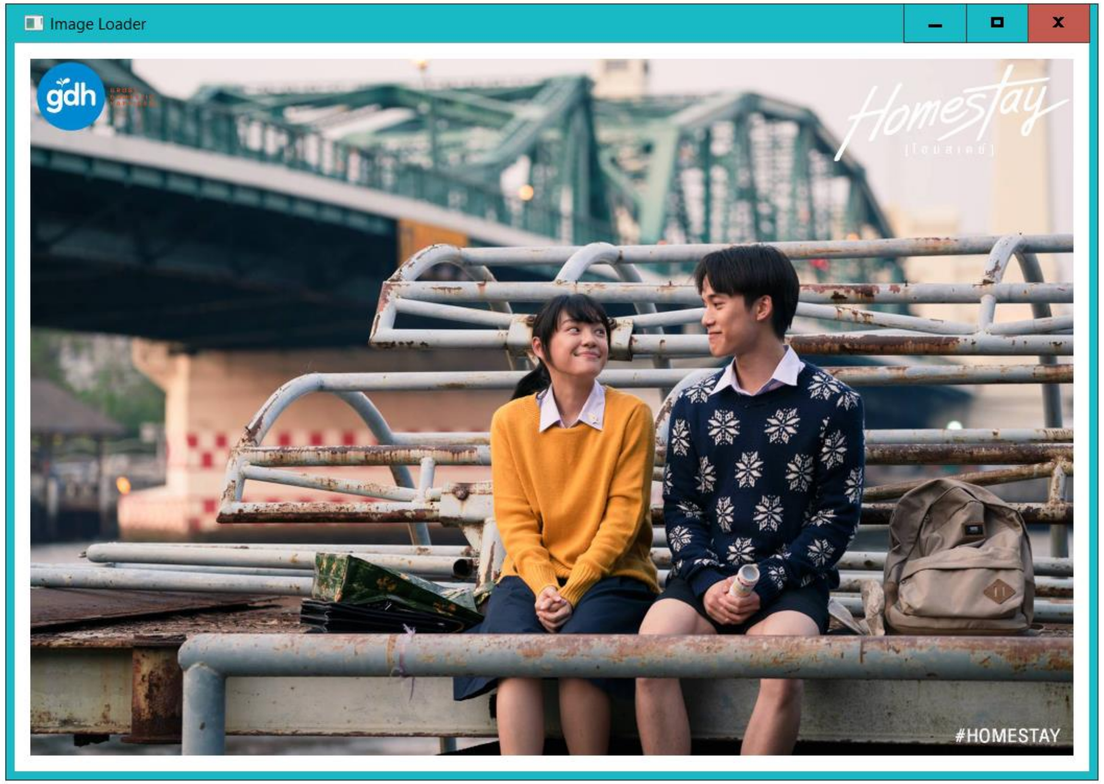
Our Image doesn’t appear anymore

Export Jar with res folder

- Let’s take a look at how we load our image
  - ImageView imageView = new ImageView(new Image("file:res/images/homestay.jpg"));
- The image must be in the same directory as our JAR
  - Let’s try again

Export Jar with res folder (cont.)

- Run -> JAVA_FX_Image/Test_Jar/2_jar_with_res_folder/run.jar
- It's work !!!

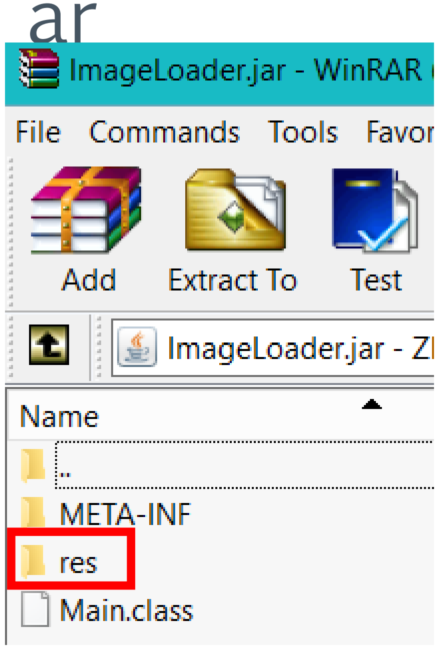


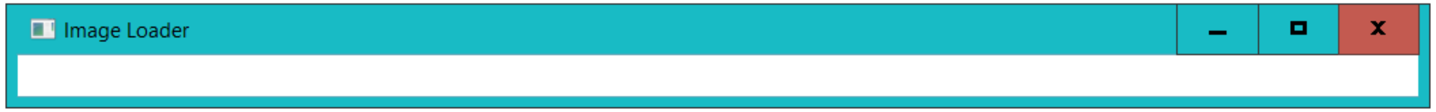
Export Jar containing res folder

- Keeping resource beside our JAR works
- But it would be better if we can store all our resources into our JAR

Export Jar containing res folder (cont.)

- Run -> JAVA_FX_Image/Test_Jar/3_jar_contain_res_folder/run.jar





Our Image still doesn't appear

Export Jar containing res folder (cont.)

- Why?
  - Because ImageView imageView = new ImageView(new Image("file:res/images/homestay.jpg"));
  - Can get resource from file only

- How to fix it?

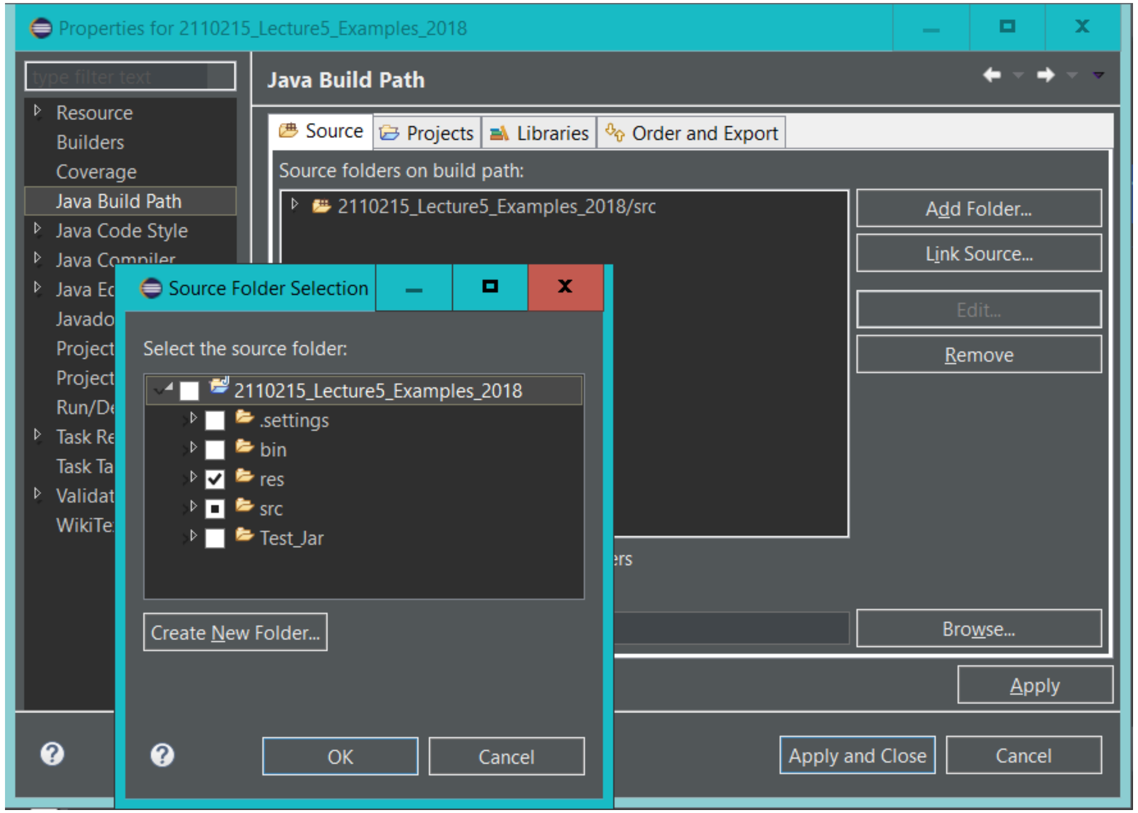
Export Jar containing res folder - ClassLoader

- Use ClassLoader to help loading our image
  - A path to our resource related to our .class file directory
- ClassLoader.*getSystemResource(String filePath)*
  - Return as URL

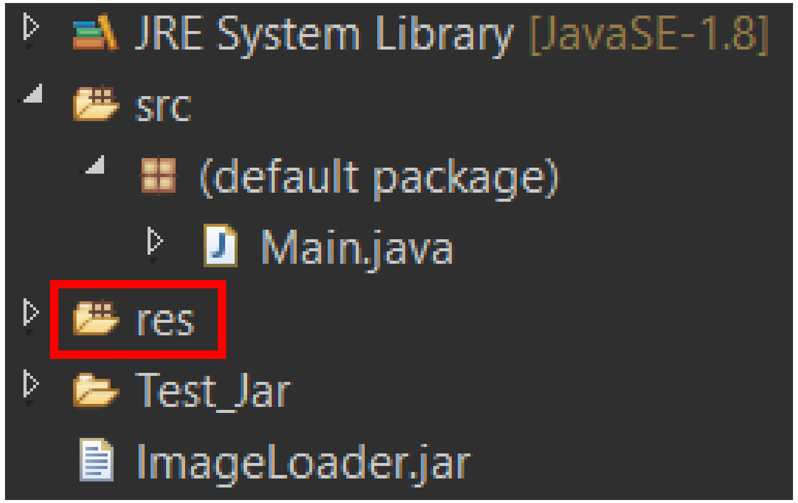
- Example:
  - String image_path = **ClassLoader.*getSystemResource("images/homestay.jpg").toString();***
  - ImageView imageView = **new ImageView(**new Image(image_path)*);*

Export Jar containing res folder - BuildPath

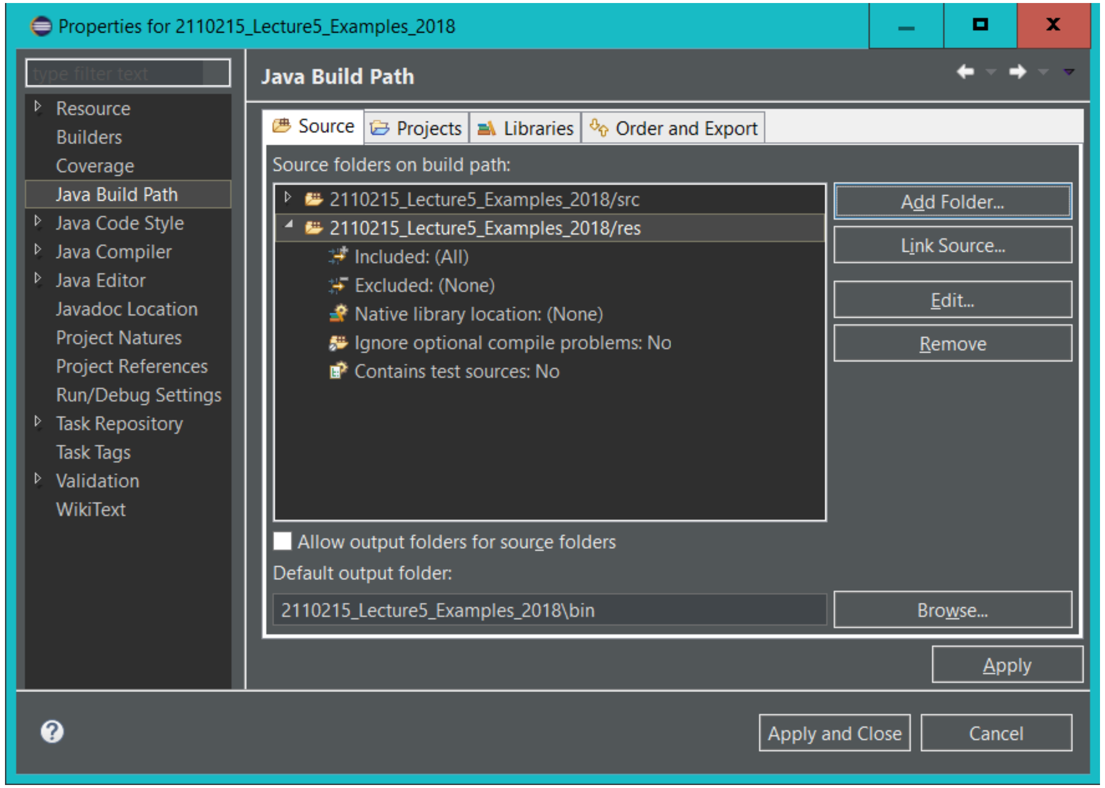
- For add resoure folder, Build Path
- -> Configure Build Path
- -> Source (Tab)
- -> Add Folder
- -> Select Folder res



Export Jar containing res folder - BuildPath



```
▷ JRE System Library [JavaSE-1.8]
◢ src
    ◢ (default package)
        ▷ Main.java
▷ res
▷ Test_Jar
  ImageLoader.jar
```



Properties for 2110215_Lecture5_Examples_2018

Java Build Path

Source folders on build path:

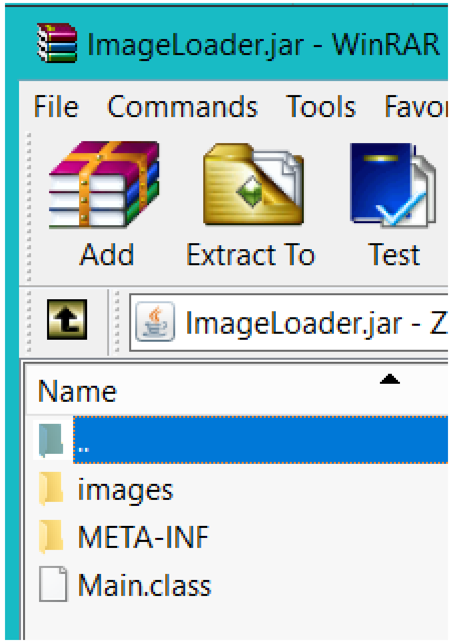
```
▷ 2110215_Lecture5_Examples_2018/src
◢ 2110215_Lecture5_Examples_2018/res
    Included: (All)
    Excluded: (None)
    Native library location: (None)
    Ignore optional compile problems: No
    Contains test sources: No
```

☐ Allow output folders for source folders

Default output folder: 2110215_Lecture5_Examples_2018\bin

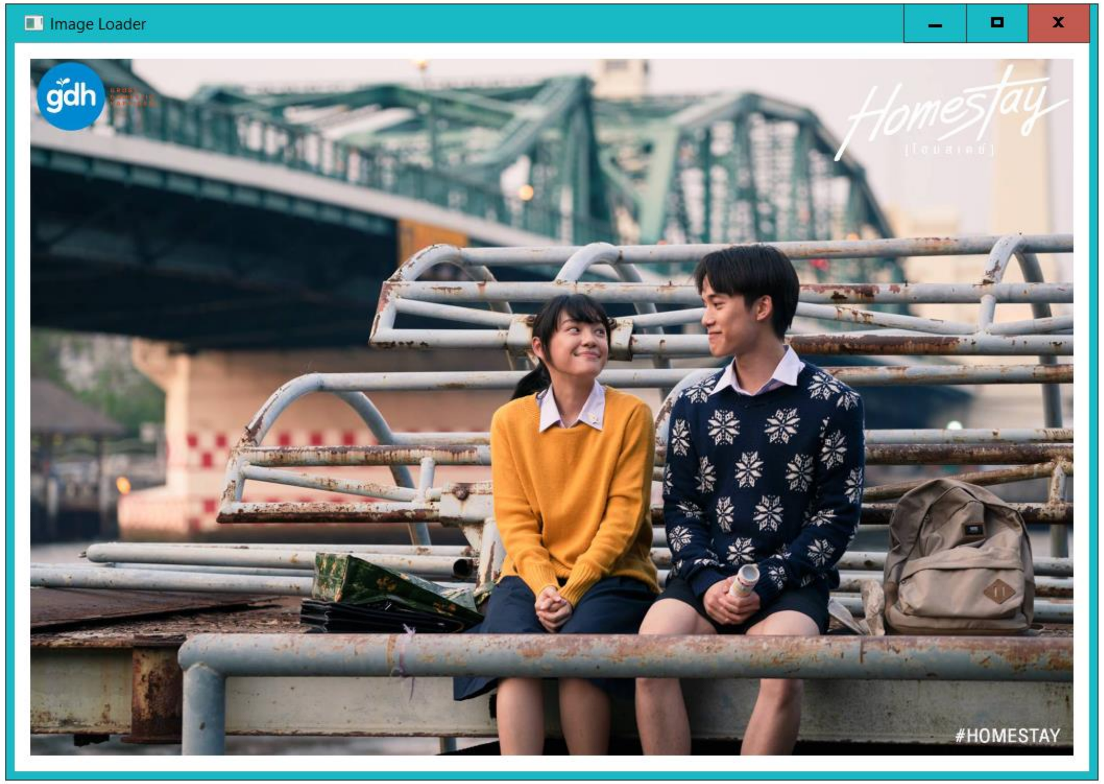
Export Jar containing res folder (cont.)

- Let try export runnable jar
- Our jar will contain res folder automatically
- No need to put our res folder in jar manually



Export Jar containing res folder (cont.)

- Run -> JAVA_FX_Image/Test_Jar/4_jar_fixed/run.jar
- This works because it read resource from our jar file.



Export Jar (cont.)

- Note
  - Some library can get resource from String : “image/homestay.jpg"
  - While some library can’t
- Using ClassLoader for getting resource for all library is more stable