Chapter 6

JavaFX Technologies

Chapter Outlines

- Introduction to JavaFX
- Nodes (Button, TextField, Label, ImageView, etc.)
- Event Handlers (Button event, Keyboard event, etc.)
- Layouts (HBox, VBox, BorderPane, GridPane, etc.)
- Web Engine (HTML, CSS, etc.)
- Media Engine (audio, video)
- Laboratory & Project

Rich Internet Applications (RIA)

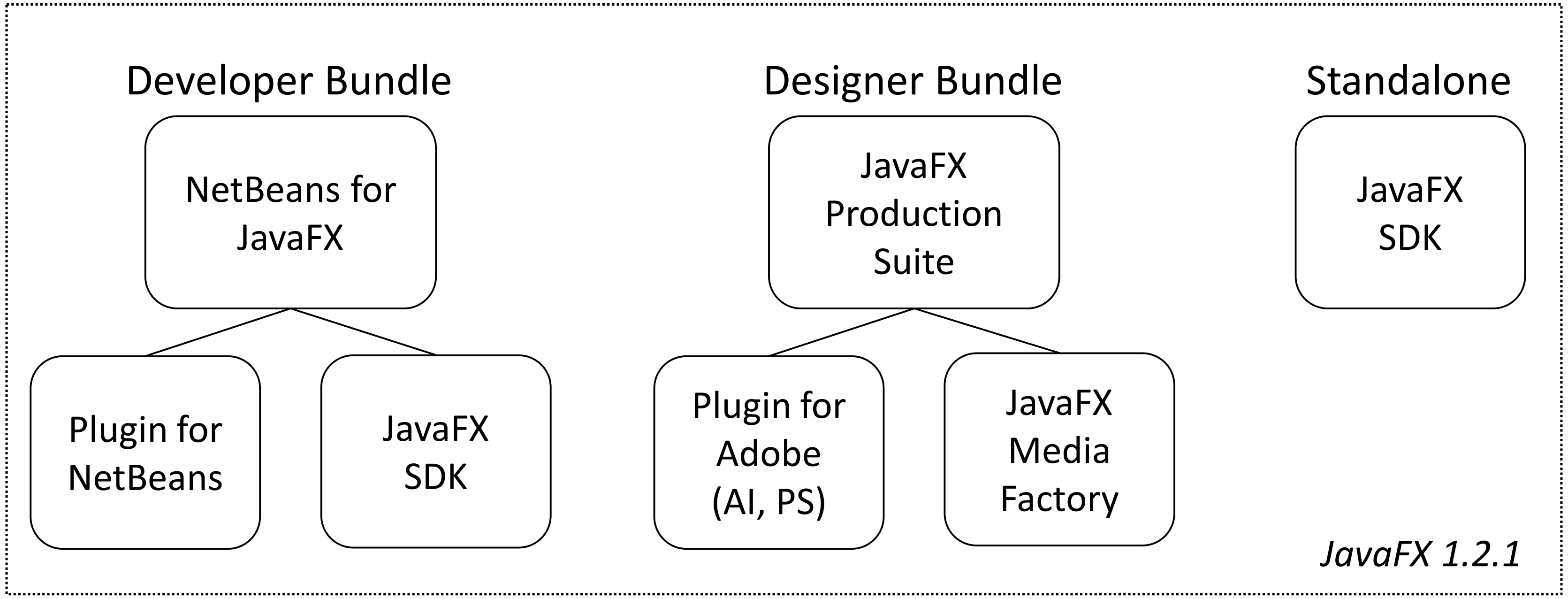
- Rich Client Applications
  - Desktop applications and platform dependent
  - Provide more functions and controls to clients
- Thin Client Applications
  - Web applications (Browser based) and cross platform
  - Many restrictions and require user authorities to install additional plugins
- Rich Internet Applications
  - Applications that have many characteristics of desktop applications and can run across a wide variety of devices

Java GUI Milestones

- Java AWT (Java Foundation Classes *JFC*)
  - Abstract Window Toolkit is the standards API for Java GUI
  - Heavyweight (platform-specific code)
  - It uses the OS native libraries to render GUI components
- Java Swing (Java 1.2 and later)
  - Lightweight (platform-independent)
  - Completely in Java for rendering GUI components
  - Only for desktop applications
- JavaFX (Java 7 and later)
  - Rich Internet Applications running purely on JVM
  - Support most of the modern web and media technologies
  - Able to run on desktop, website, mobile devices, IP TV, etc.

About JavaFX

- Sun launched F3 (Form Follows Function) as JavaFX platform at 2007
- JavaFX 1.0 was released at December 2008

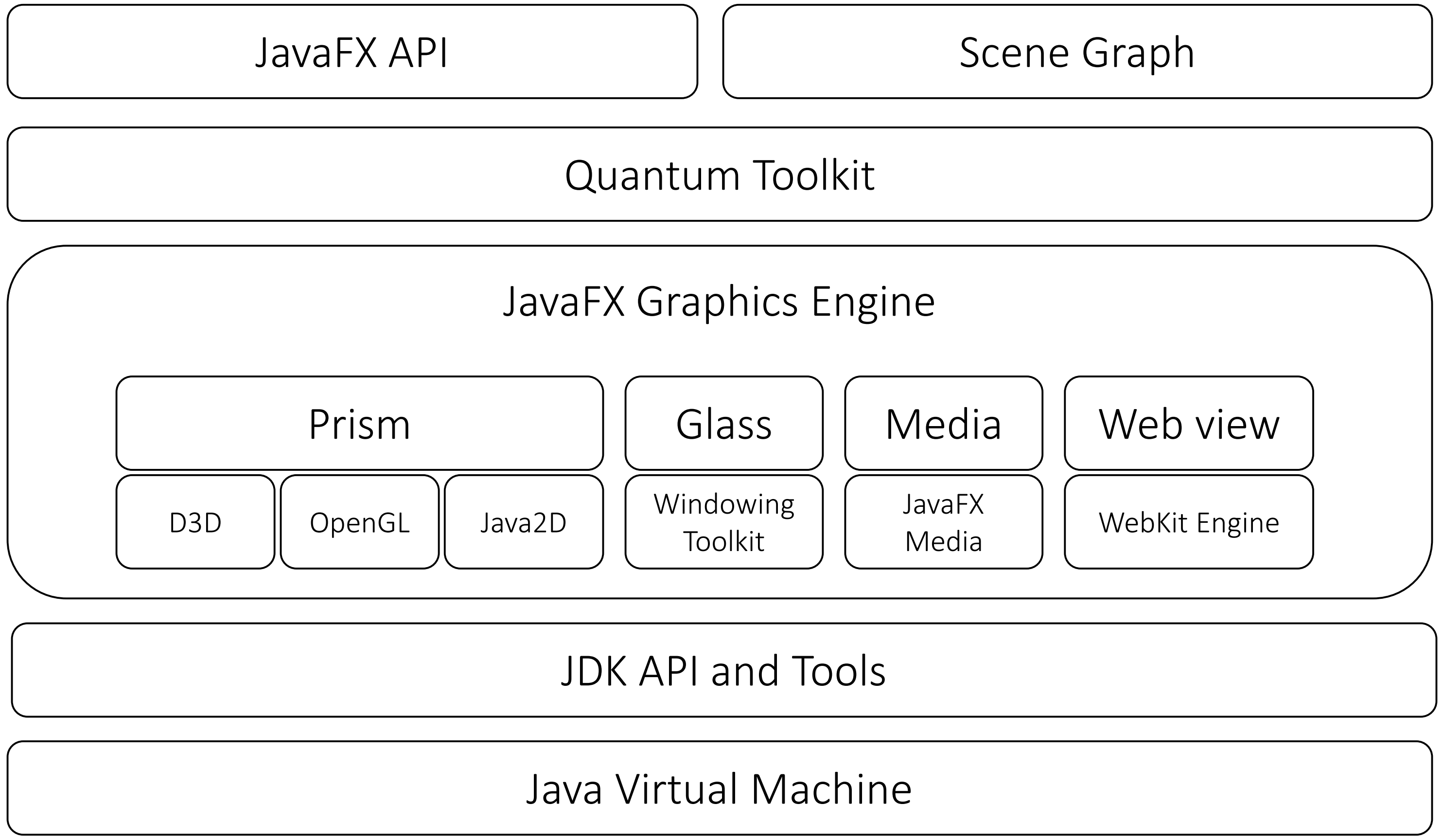


JavaFX Version

- JavaFX 1.0 to JavaFX 1.3.1
  - Uses JavaFX Script which is compiled to Java byte code
- JavaFX 2.0 to JavaFX 2.2
  - Use native Java library to write pure native Java code
- JavaFX 8
  - Embedded in Java SE 8 by Oracle
  - Attempt to replace Swing
- JavaFX 9
  - It is part of Java JRE/SDK 9 (coming soon…)

JavaFX Architecture

- It is able to run on different devices



First JavaFX Program

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;


public class HelloJavaFX extends Applicaiton {
 /**
  * @param args the command line arguments
  */
 public static void main(String[] args) {
   // Execute the JavaFX Rich Internet Application
   Application.launch(args);
 }

```

First JavaFX Program (cont.)

```
  @Override
  public void start(Stage stage) {
    stage.setTitle("Hello World!");
    Group root = new Group();
    Scene scene = new Scene(root, 640, 480);
    Button btn = new Button("Hello World!");
    btn.setLayoutX(100);
    btn.setLayoutY(80);
    btn.setOnAction((ActionEvent event) -> {
      System.out.println("Hello World!");
    });
    root.getChildren().add(btn);
    stage.setScene(scene);
    stage.show();
  }
}

```

Stage and Scene

- The JavaFX designers model things on the idea of a theater or a play in which actors perform in front of an audience
- *Stage* is a screen (a window, a monitor)
- *Scene* is a page (panel, pane, etc.)
- Players act different scenes on a stage
- A scene can contain many JavaFX components, and they are known as the *node* objects
  - layouts (HBox, VBox, FlowPane, BorderPane, GridPane, etc.)
  - Button, Label, TextField, etc.

javafx.application

- JavaFX programs will extend *javafx.application.Application* class which provide the lifecycle functions for the application
    - initializing, launching, starting, and stopping
- All JavaFX programs use the following template

```
public class HelloJavaFX extends Applicaiton {
  public static void main(String[] args) {
    // Program’s main thread
    Application.launch(args);
  }

  @Override
  public void start(Stage stage) {
    // JavaFX application thread
  }
}
```

Threading

- When the *launch()* method is executed, it will invoke the *start()* method to begin the application
- The program enters a ***ready*** state and it will continue run on the JavaFX application thread instead of the main thread
- There are different types of thread
  - Initial Thread: execute the initial application code
  - Event Dispatch Thread (EDT): where all event-handling code is executed (button, keyboard, etc. events)
  - Worker Thread: background threads to execute time-consuming tasks
- When components become ***realized*** (the paint() method is called to become visible), they will be executed in the EDT

Nodes

- Node is a fundamental base class for all scene graph nodes to be rendered
- Some of the commonly used nodes are
  - javafx.scene.control.Button

    https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Button.html
  - javafx.scene.control.Label

    https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Label.html
  - javafx.scene.control.TextField

    https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TextField.html
  - javafx.scene.control.ImageView

    https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/ImageView.html

Creating Nodes

- They are easy to use

```
…
Button btn1 = new Button();
btn1.setText("Function One");
Button btn2 = new Button("Function Two");

Label label1 = new Label();
label1.setText("Label One");
Label label2 = new Label("Label Two");

TextField tf1 = new TextField();
tf1.setText("Textfield One");
TextField tf2 = new TextField("Textfield Two");
…
```

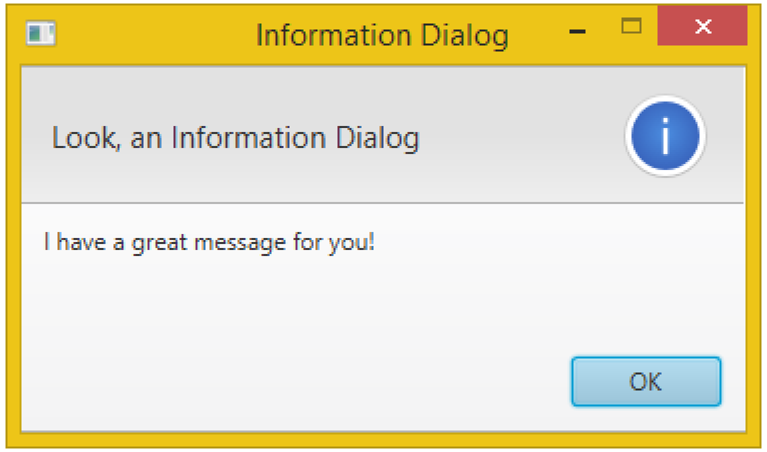
Alert (Dialog box)

- The *Alert* class inherits the *Dialog* class, and provides support for a number of pre-built dialog types that can be easily shown to users
- Different types of *Alert*
  - Information, Warning, Error, Confirmation, etc.
- An *Alert* class contains different components
  - Title, Header, Content, Buttons
- Blocking / Non-Blocking
  - showAndWait() / show()

Information Type

- Use to inform a piece of information

```
Alert alert = new Alert(AlertType.INFORMATION);
alert.setTitle("Information Dialog");
alert.setHeaderText("Look, an Information Dialog");
alert.setContentText("I have a great message for you!");
alert.showAndWait();
```



Information Dialog

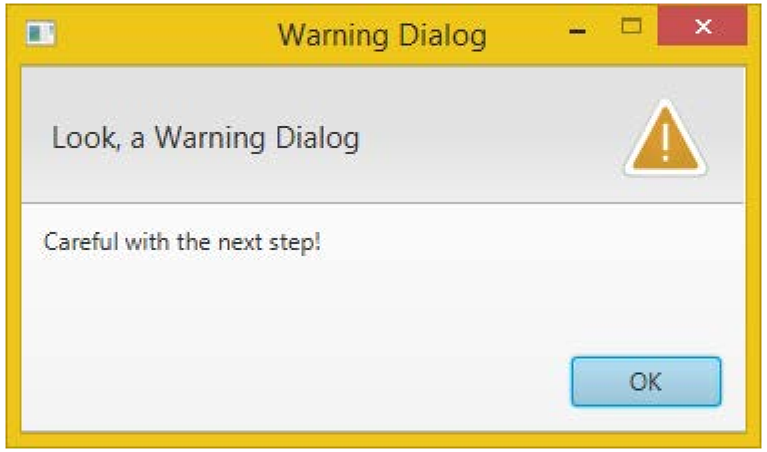
Look, an Information Dialog

I have a great message for you!

Warning Type

- Use to warn about some fact or action

```
Alert alert = new Alert(AlertType.WARNING);
alert.setTitle("Warning Dialog");
alert.setHeaderText("Look, a Warning Dialog");
alert.setContentText("Careful with the next step!");
alert.showAndWait();
```



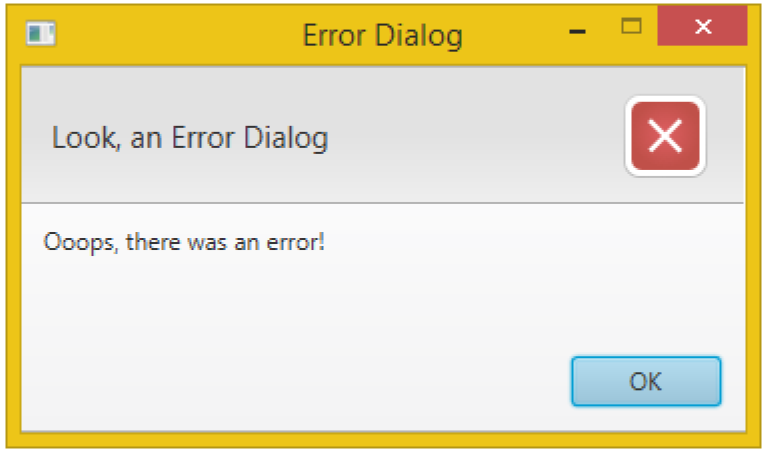
Look, a Warning Dialog

Careful with the next step!

Error Type

- Use to report that something has gone wrong

```
Alert alert = new Alert(AlertType.ERROR);
alert.setTitle("Error Dialog");
alert.setHeaderText("Look, an Error Dialog");
alert.setContentText("Ooops, there was an error!");
alert.showAndWait();
```


Error Dialog

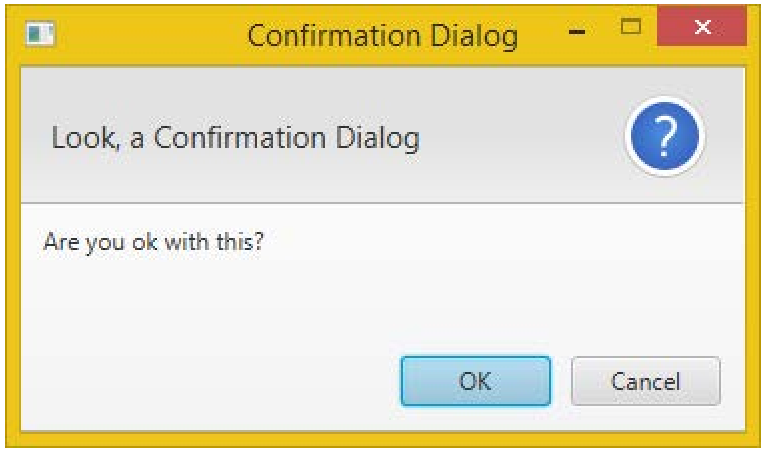
Look, an Error Dialog

Ooops, there was an error!

Confirmation Type

- Use to seek confirmation from the users

```
Alert alert = new Alert(AlertType.CONFIRMATION);
alert.setTitle("Confirmation Dialog");
alert.setHeaderText("Look, a Confirmation Dialog");
alert.setContentText("Are you ok with this?");
Optional<ButtonType> result = alert.showAndWait();
if (result.get() == ButtonType.OK) { /* handle user clicks OK */ }
else { /* handle user clicks CANCEL */ }
```


Confirmation Dialog

Look, a Confirmation Dialog

Are you ok with this?

Blocking vs Non-Blocking

- After the *Alert* is shown, the program will be paused and wait for the respond from the user
- The code after *Line-15* will not be executed (*Blocking*) until the user has closed the *Alert*
- alert.show() will continue to execute (*Non-Blocking*) the rest of the codes

```
11.    Alert alert = new Alert(AlertType.INFORMATION);
12.    alert.setTitle("Information Dialog");
13.    alert.setHeaderText("Look, an Information Dialog");
14.    alert.setContentText("I have a great message for you!");
15.    alert.showAndWait();
16.    System.out.println("Will be executed after the user has made the response");
```

ImageView

- It is a *Node* used for painting images loaded with the *Image* class
- It cannot ***render*** animation picture (GIF, PNG, etc.)
- It supports relative and absolute paths (full package name), and the absolute path is safe to use
- It provides plenty of effects to modify the images
  - Blend, Bloom, BoxBlur, ColorAdjust, ColorInput, DisplacementMap, DropShadow, GaussianBlur, Glow, ImageInput, InnerShadow, Lighting, MotionBlur, PerspectiveTransform, Reflection, SepiaTone, Shadow

  https://docs.oracle.com/javase/8/javafx/api/javafx/scene/effect/Effect.html

ImageView Example

- Put the image files to the folder (package) inside the `src` folder
  - Image File: logo.png
  - Package: ipm.esap.comp221.media
- Use the *Image* class to load the image file (logo.png)
- Use the *ImageView* to pack the loaded image and apply your favorite effects to this image

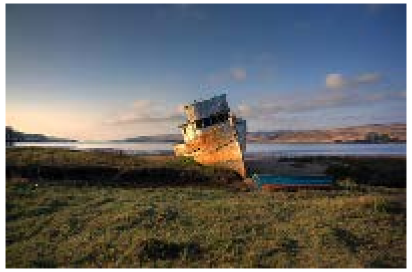
```
package ipm.esap.comp221;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
  …
  Image image = new Image("/ipm/esap/comp221/media/logo.png");
  ImageView imageView = new ImageView(image);
  StackPane root = new StackPane();
  root.getChildren().add(imageView);
 …
```

Glow Effect

- The following example illustrate the glow effect

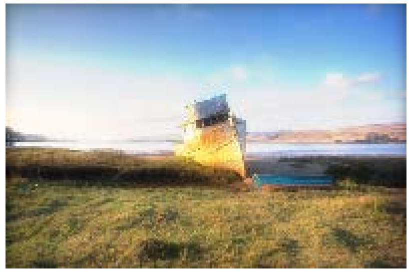
```
package ipm.esap.comp221;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.effect.Glow;
  …
  Image image = new Image("/ipm/esap/comp221/media/boat.png");
  ImageView imageView = new ImageView(image);
  imageView.setEffect(new Glow(0.8));
 …

```



Original picture

Glow effect



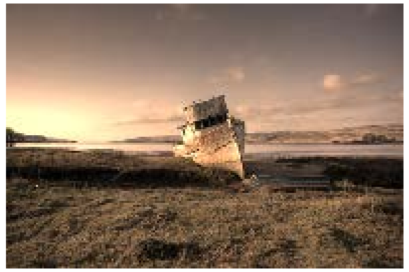
SepiaTone Effect

- The following example illustrate the sepia tone effect

```
package ipm.esap.comp221;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.effect.Glow;
 …
 SepiaTone sepiaTone = new SepiaTone();
 sepiaTone.setLevel(0.7);
 Image image = new Image("/ipm/esap/comp221/media/boat.png");
 ImageView imageView = new ImageView(image);
 imageView.setEffect(sepiaTone);
…
```

Original picture

Sepia tone effect



Changing an image

- Replace an image or show an animating image in Swing is not an easy task. Since the painted image is in a ***ready*** state, thread objects are required to use for repainting the image
- *ImageView* class provides a *setImage()* method to change the image easily

```
…
  String[] pics = new String[] { "picture01.png", "picture02.png" };
  Image image = new Image(MEDIA_FOLDER + pics[flag]);
  ImageView imageView = new ImageView(image);
  Button btnChange = new Button("Next");
  btnChange.setOnAction((ActionEvent event) -> {
    imageView.setImage(new Image(MEDIA_FOLDER + pics[++flag % pics.length]));
  });
…
```

Handling Events

- There are two ways to handle the fire button event
  - Anonymous Class / Lambda Expression
  - A handler method

```
Button btn = new Button("Hello World!");
btn.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent event) {
    System.out.println("Hello World!");
    // Implement the logic here
  }
});
```

```
Button btn = new Button("Hello World!");
btn.setOnAction((ActionEvent event) -> {
  System.out.println("Hello World!");
  // Implement the logic here
});
```

A handler method

```
  Button btn1 = new Button("Button One");
  btn1.setOnAction(fireButton());
  Button btn2 = new Button("Button Two");
  btn2.setOnAction(fireButton());
  …
}
public EventHandler<ActionEvent> fireButton() {
  return new EventHandler<ActionEvent>() {
    public void handle(ActionEvent ae) {
      if (ae.getSource() instanceof Button) {
        if (ae.getSource().equals(btn1)) { /* logic for button 1 */ }
        else if (ae.getSource().equals(btn2)) {/* logic for button 2 */ }
        else { /* don’t handle */ }
      }
    }
  };
}
```

Keyboard Events

- Monitors the key typed by users

```
public void start(Stage primaryStage) {
  Label message = new Label("Type here: ");
  StackPane root = new StackPane();
  root.setAlignment(Pos.CENTER);
  root.getChildren().add(message);
  Scene scene = new Scene(root, 320, 240);
  scene.setOnKeyPressed((KeyEvent ke) -> {
    message.setText("The key <" + ke.getCode() + "> is Typed.");
    if (ke.getCode() == KeyCode.ESCAPE) {
      System.exit(0);
    }
  });
  primaryStage.setTitle("Keyboard Event");
  primaryStage.setScene(scene);
  primaryStage.show();
}
```

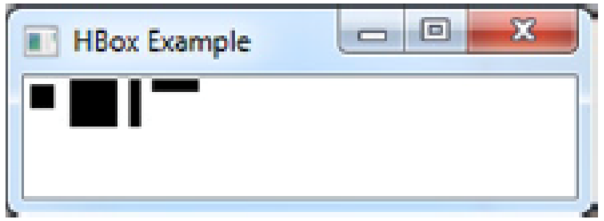
Layouts

- Scene graph provides plenty of layouts to display UI
- Layout lines them up from top left to bottom right
- It is designed to pack many JavaFX nodes together
- It can organize the nodes (textfield, button, label, etc.) into a display area, and some of the commonly used layouts are
  - java.scene.layout.HBox
  - java.scene.layout.VBox
  - java.scene.layout.BorderPane
  - java.scene.layout.GridPane

The HBox Layout

- HBox represents the horizontal box
- It lines up the nodes from left to right horizontally

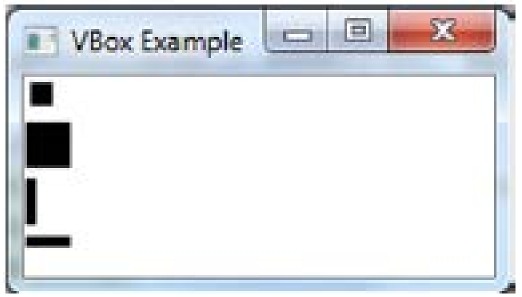
```
HBox hbox = new HBox(5);
Rectangle r1 = new Rectangle(10, 10);
Rectangle r2 = new Rectangle(20, 20);
Rectangle r3 = new Rectangle(5, 20);
Rectangle r4 = new Rectangle(20, 5);
HBox.setMargin(r1, new Insets(2, 2, 2, 2)); // spacing around r1
hbox.getChildren().addAll(r1, r2, r3, r4);
```



The VBox Layout

- VBox represents the vertical box
- It lines up the nodes from top to bottom vertically

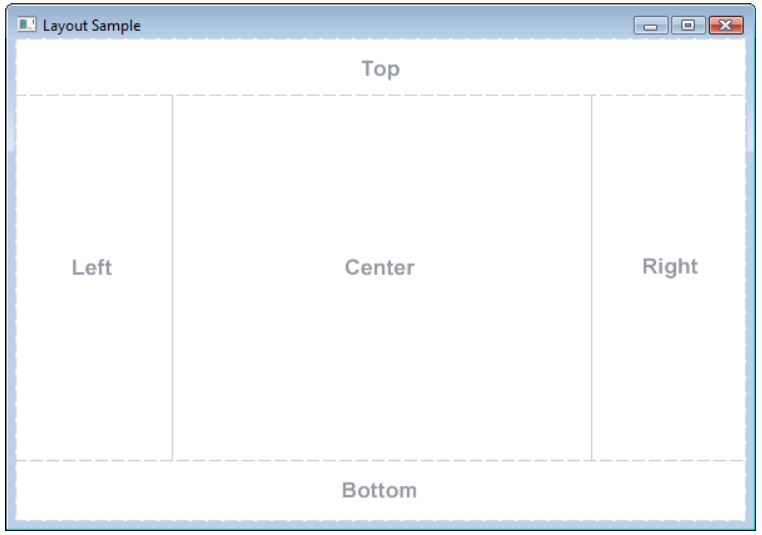
```
VBox vbox = new VBox(5);
Rectangle r1 = new Rectangle(10, 10);
Rectangle r2 = new Rectangle(20, 20);
Rectangle r3 = new Rectangle(5, 20);
Rectangle r4 = new Rectangle(20, 5);
VBox.setMargin(r1, new Insets(2, 2, 2, 2)); // spacing around r1
vbox.getChildren().addAll(r1, r2, r3, r4);
```



The BorderPane Layout

- It allows child nodes to be placed in a top, bottom, left, right, or center region
- Because each region can only have a node, we will often use other layouts (HBox, VBox, etc.) to group the nodes first

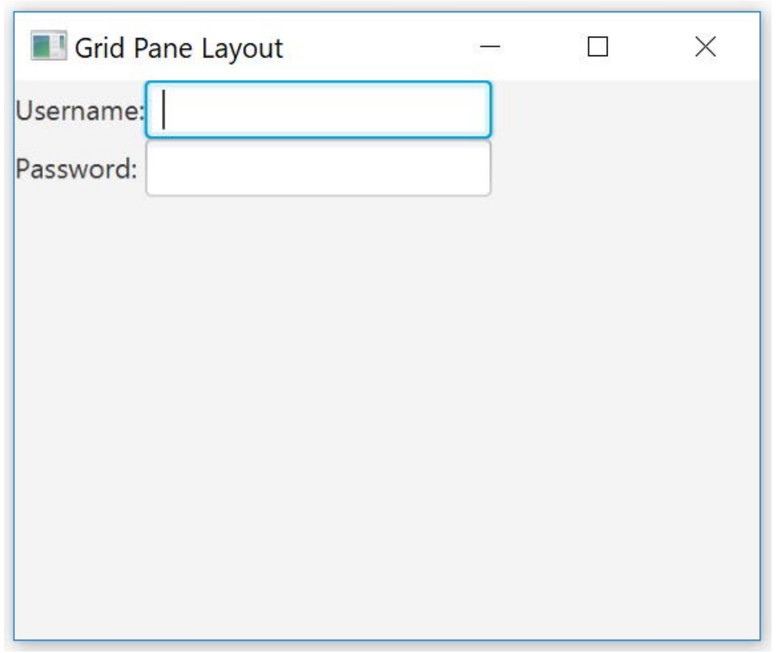
```
Label body = new Label("Hello World!");
Button btn1 = new Button("Submit");
Button btn2 = new Button("Back");
HBox hbox = new HBox();
hbox.getChildren().addAll(btn1, btn2);
BorderPane pane = new BorderPane();
pane.setCenter(body);
pane.setBottom(hbox);
```



The GridPane Layout

- It provides a grid pattern to align the nodes. It is liked a table to have rows and columns

```
GridPane pane = new GridPane();
pane.add(new Label("Username:"), 1, 1); // column=1, row=1
pane.add(new TextField(), 2, 1);        // column=2, row=1
pane.add(new Label("Password:"), 1, 2); // column=1, row=2
pane.add(new TextField(), 2, 2);        // column=2, row=2
```



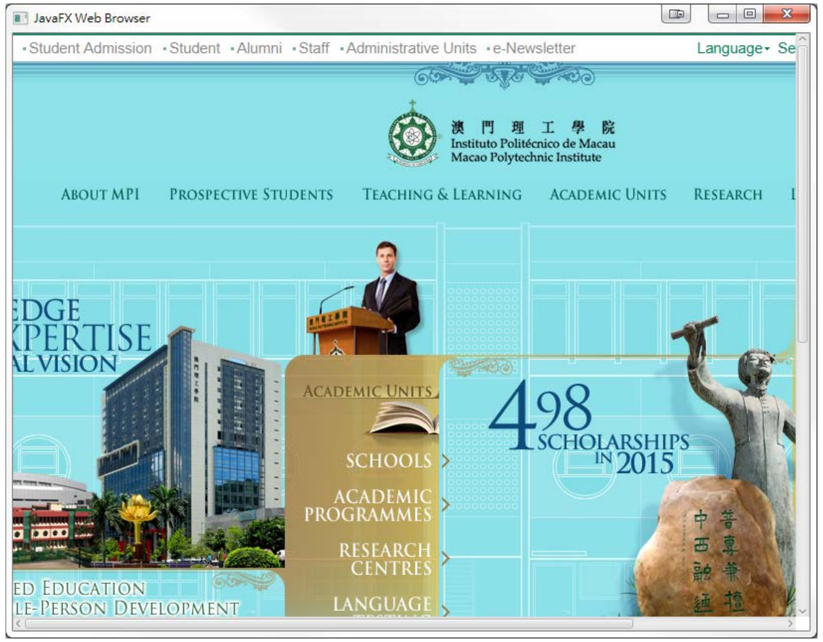
Web Engine

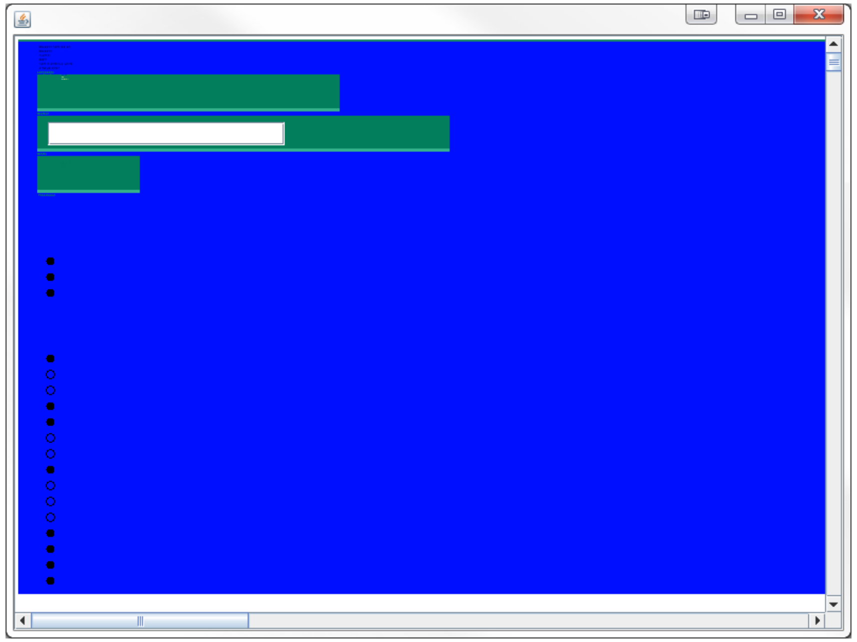
- JavaFX uses the famous WebKit (open-source API) layout engine to handle web content (HTML)
  - This API is used by Safari, Chrome, Amazon’s Kindle devices, etc.
- It allows JavaFX applications to support multimedia files (JS, HTML5, CSS, SVG, Canvas, Media, XML, etc.) that Swing can’t

```
public void start(Stage primaryStage) {
  WebView browser = new WebView();
  WebEngine webEngine = browser.getEngine();
  webEngine.load("http://www.ipm.edu.mo/");
  StackPane root = new StackPane();
  root.getChildren().add(browser);
  Scene scene = new Scene(root, 800, 600);
  primaryStage.setScene(scene);
  primaryStage.show();
}
```

Swing Version

- Browse the MPI website with JavaFX and Swing
- Swing use *JEditorPane* class to render HTML content
- The result on the right-hand side is terrible





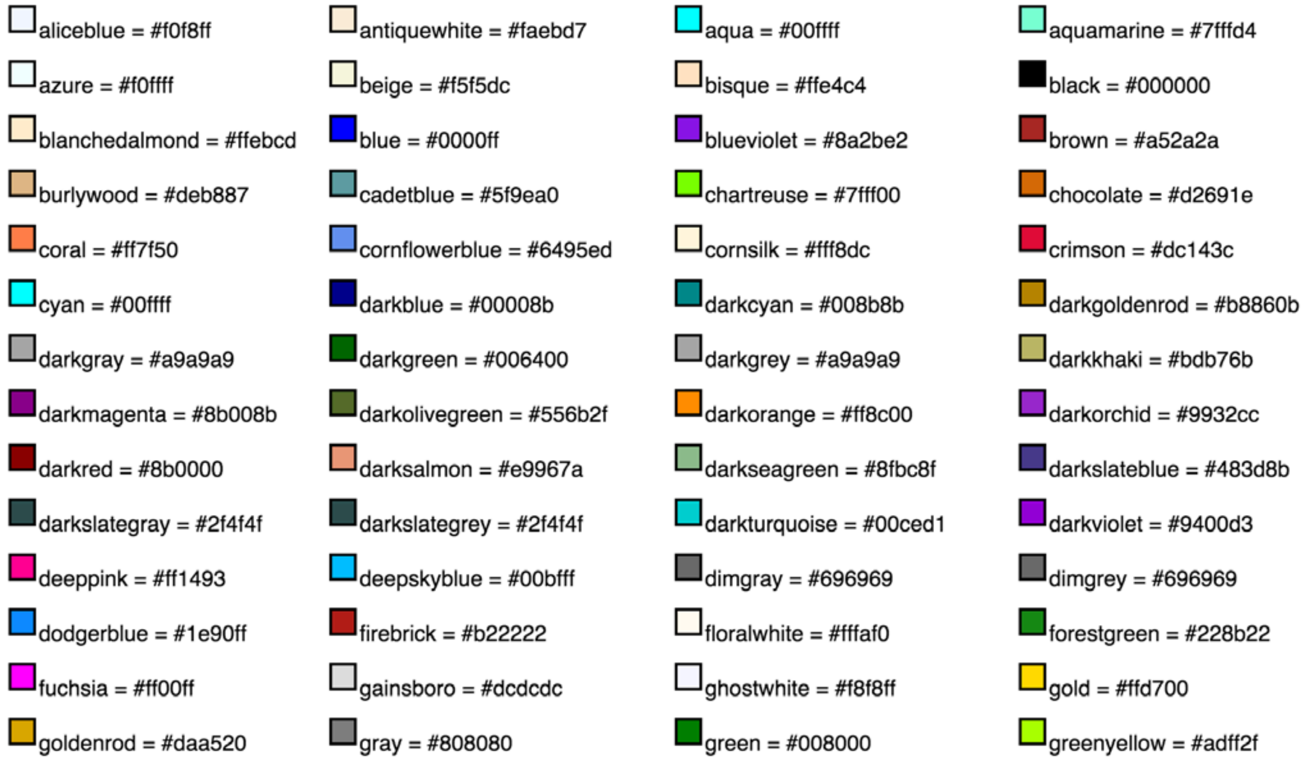
JavaFX CSS

- Styling the Java UI with JavaFX CSS (Cascading Style Sheets) is easy and elegant
  - Swing doesn’t support CSS
- JavaFX CSS is based on the W3C CSS standards with the prefix “-fx-”

  https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html
- There are mainly two ways to use JavaFX CSS
  - Write the styles directly inside the Java programs
  - Write a CSS file for the Java programs to read pretty much like developing a web page

JavaFX CSS colors

- CSS supports a bunch of named constant colors. They are mapped to the standard RGB colors

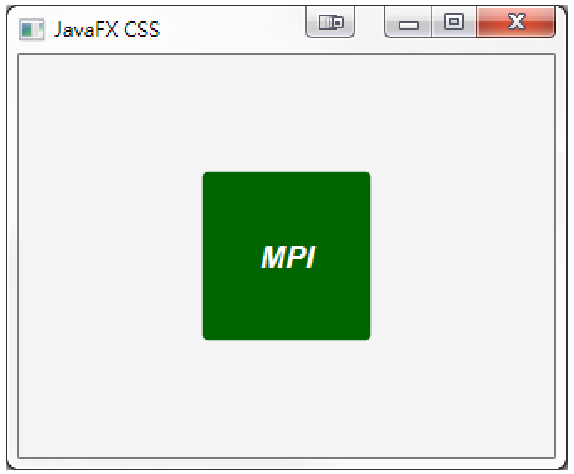


⬜ aliceblue = #f0f8ff	⬜ antiquewhite = #faebd7	🟦 aqua = #00ffff	🟩 aquamarine = #7fffd4
⬜ azure = #f0ffff	⬜ beige = #f5f5dc	⬜ bisque = #ffe4c4	⬛ black = #000000
⬜ blanchedalmond = #ffebcd	🟦 blue = #0000ff	🟪 blueviolet = #8a2be2	🟫 brown = #a52a2a
🟫 burlywood = #deb887	🟦 cadetblue = #5f9ea0	🟩 chartreuse = #7fff00	🟧 chocolate = #d2691e
🟧 coral = #ff7f50	🟦 cornflowerblue = #6495ed	⬜ cornsilk = #fff8dc	🟥 crimson = #dc143c
🟦 cyan = #00ffff	🟦 darkblue = #00008b	🟦 darkcyan = #008b8b	🟫 darkgoldenrod = #b8860b
⬜ darkgray = #a9a9a9	🟩 darkgreen = #006400	⬜ darkgrey = #a9a9a9	🟨 darkkhaki = #bdb76b
🟪 darkmagenta = #8b008b	🟩 darkolivegreen = #556b2f	🟧 darkorange = #ff8c00	🟪 darkorchid = #9932cc
🟥 darkred = #8b0000	🟧 darksalmon = #e9967a	🟩 darkseagreen = #8fbc8f	🟦 darkslateblue = #483d8b
⬛ darkslategray = #2f4f4f	⬛ darkslategrey = #2f4f4f	🟦 darkturquoise = #00ced1	🟪 darkviolet = #9400d3
🟥 deeppink = #ff1493	🟦 deepskyblue = #00bfff	⬛ dimgray = #696969	⬛ dimgrey = #696969
🟦 dodgerblue = #1e90ff	🟥 firebrick = #b22222	⬜ floralwhite = #fffaf0	🟩 forestgreen = #228b22
🟪 fuchsia = #ff00ff	⬜ gainsboro = #dcdcdc	⬜ ghostwhite = #f8f8ff	🟨 gold = #ffd700
🟨 goldenrod = #daa520	⬛ gray = #808080	🟩 green = #008000	🟩 greenyellow = #adff2f

Button Style

- Changing the styles of a button

```
Button btn = new Button("MPI");
btn.setMinWidth(100);
btn.setMinHeight(100);
btn.setStyle("-fx-background-color: darkgreen; -fx-text-fill: white; " +
        "-fx-font-family: Arial; -fx-font-size: 18; " +
        "-fx-font-weight: bold; -fx-font-style: italic; ");
```



Using a CSS file

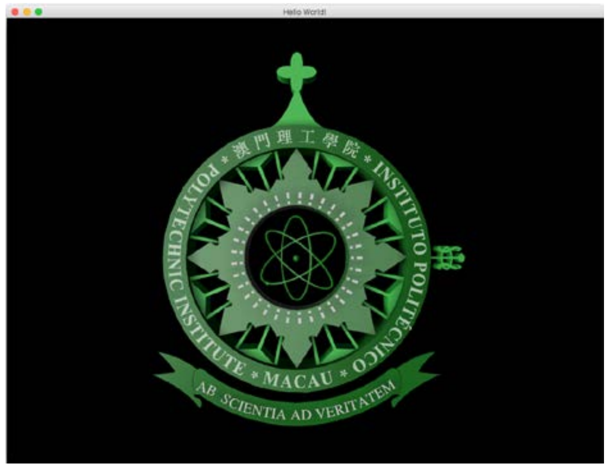
- Create a CSS file (style.css) in the same folder
- Import the style sheet as in *Line-16*

```
01.   package ipm.esap.comp221;
02.   …
10.     public void start(Stage primaryStage) {
11.       Label message = new Label("Hello");
12.       StackPane root = new StackPane();
13.       root.setAlignment(Pos.CENTER);
14.       root.getChildren().add(message);
15.       Scene scene = new Scene(root, 320, 240);
16.       scene.getStylesheets().add("/ipm/esap/comp221/style.css");
17.       primaryStage.setTitle("JavaFX CSS");
18.       primaryStage.setScene(scene);
19.       primaryStage.show();
20.     }
```

CSS File

- The CSS file changes the background of the scene

```
/* JavaFX CSS File */
.root {
  -fx-background-image: url('/ipm/esap/comp221/media/Background.jpg');
  -fx-background-position: center center;
  -fx-background-repeat: stretch;
  -fx-background-color: black;
}
```



CSS for all buttons

- Use the reserved word to set the style for all buttons

```
/* JavaFX CSS File */
.button {
  -fx-background-color: darkgreen;
  -fx-text-fill: white;
  -fx-font-family: Arial;
  -fx-font-size: 18;
}
```

- It will automatically apply the style (as the default style) to a new born button

```
 Button btn = new Button("Submit");
 btn.setMinWidth(100);
 btn.setMinHeight(100);
```

Define a style in CSS

- Use the reserved word to set the style for Label

```
/* JavaFX CSS File */
.button {
  -fx-background-color: darkgreen; -fx-text-fill: white;
  -fx-font-family: Arial; -fx-font-size: 18;
}
.myStyle {
  -fx-font-weight: bold;
  -fx-font-style: italic;
}
```

- A button can override the default *button* style

```
Button btn = new Button("Submit");
btn.setMinWidth(100);
btn.setMinHeight(100);
btn.getStyleClass().add("myStyle");
```

Media Engine

- JavaFX provides a media-rich API for playing audio and video. It supports many media formats (AAC, mp3, H.264, mp4, etc.)
  https://docs.oracle.com/javase/8/javafx/api/javafx/scene/media/package-summary.html
- It is cross-platform to support various devices (tablet, tv, mobile, music player, etc.) to play multimedia
- It uses the *Media* class to load the multimedia files first. Then, it uses the *MediaPlayer* class to control them with the following methods
  - play(), pause(), mute(), stop(), setVolume(double value), etc.

  https://docs.oracle.com/javase/8/javafx/api/javafx/scene/media/MediaPlayer.html

Music Player Example

```
package ipm.esap.comp221;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
…
public class MusicPlayer extends Application {
  public void start(Stage primaryStage) {
    String song = "/ipm/esap/comp221/media/music.mp3";
    Media media = new Media(Class.class.getResource(song).toString());
    MediaPlayer mediaPlayer = new MediaPlayer(media);
    mediaPlayer.setAutoPlay(true);
    Scene scene = new Scene(new Label("Music Box"), 200, 150);
    primaryStage.setTitle("Music Player");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) { launch(args); }
}
```

Summary

- JavaFX is the latest technologies for replacing the Java Swing to build graphical user interfaces
- It provides better solutions to present web pages and multimedia files
- It is able to run on different devices and platforms
- There are software to convert the JavaFX projects into mobile applications (Android, iOS, Window Mobile, etc.)
  - JavaFXPorts
  - Gradle